AFIT/DS/ENG/97-01

Dissertation:

Autonomous Construction of Multi Layer Perceptron Neural Networks

DISSERTATION

Thomas F. Rathbun
Captain, USAF

AFIT/DS/ENG/97-01

DTIC QUALITY INSPECTED 3

Approved for public release; distribution unlimited

19970708 115

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

DTIC QUALITY INSPECTED 3

AFIT/DS/ENG/97-01

Dissertation:
Autonomous Construction of Multi Layer Perceptron Neural Networks

DISSERTATION

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy in Computer Engineering

Thomas F. Rathbun, B.S.C.S.E., M.S.Cp.E.
Captain, USAF

June, 1997

Approved for public release; distribution unlimited

AFIT/DS/ENG/97-01

Autonomous Construction of Multi Layer Perceptron Neural Networks

Thomas F. Rathbun, B.S., M.S.

Captain, USAF

Approved:

Dr. Steven K. Rogers, Committee Chairman

Dr. Martin P. Desimio, Committee Member

Mark E. Oxley

Dr. Mark E. Oxley, Committee Member

Dr. Matthew Kabrisky, Committee Member

Anthony N Palazotto

Dr. Anthony N. Palazotto, Dean's Representative

Dr. Robert A. Calico, Jr., Dean

Acknowledgements

I thank my wife for her facilitation of my research effort. As my partner she is ever supportive, constantly reassuring, and always sacrificing. The completion of this effort is as much a tribute to her as it is to me. I thank my children for being children. The juxtaposition of the seriousness of education and playfulness at home accentuate the joy of childhood and importantance of never fully growing up.

Thomas F. Rathbun

There once was a wise old owl;
The more he heard the less he said;
The less he said the more he heard;
Why aren't we all like that wise old owl.

Anonymous

Table of Contents

Page

Acknowledgements iii

List of Figures vii

List of Tables 10

Abstract 11

I. Introduction 1-1
  1.1 Historical Background 1-1
  1.2 Problem Statement and Scope 1-2
  1.3 Dissertation Organization 1-4

II. Constructing Single Hidden Layer MLPs 2-1
  2.1 Introduction 2-1
  2.2 Ho-Kashyap Method 2-3
  2.3 Training Hidden Layer Nodes 2-5
    2.3.1 Selecting Class Pairs 2-6
    2.3.2 Selecting Inter-class Data Pairs 2-7
    2.3.3 Selecting Class Neighborhoods 2-7
    2.3.4 Testing for Separability 2-10
  2.4 Training Output Layer Weights 2-12
  2.5 Iteration of HLN Training 2-13
  2.6 Relaxation of HLNs 2-18
  2.7 Test Results For Single Hidden Layer MLPs (SL-MICA) . . . 2-20
    2.7.1 Spiral 2-22
    2.7.2 Mesh 2-23
    2.7.3 Iris 2-26

Page

2.7.4 OCR 2-26

2.7.5 XOR 2-28

2.7.6 ATM Access Control 2-29

2.7.7 Testing Summary 2-30

2.8 Conclusions 2-31

III. Theoretical Basis for MICA 3-1

3.1 Introduction 3-1

3.2 Separation Proof 3-1

3.3 Bayes Optimal Derivation 3-7

3.4 Bayes Bounding Example 3-10

3.4.1 Bayes Bound 3-10

3.4.2 Comparison with SL-MICA 3-10

3.5 Conclusions 3-10

IV. Construction of Multiple Hidden-Layer MLPs 4-1

4.1 Multi Layer MICA 4-1

4.2 Classification Proof for ML-MICA 4-2

4.3 Bayes Optimal Proof for ML-MICA 4-3

4.4 Test Results for ML-MICA 4-3

4.4.1 Iris 4-3

4.4.2 Mesh 4-4

4.4.3 OCR 4-4

4.4.4 XOR 4-6

4.4.5 ATM Access Control 4-7

4.5 SL-MICA verses ML-MICA 4-7

4.6 Test Results for MICA 4-11

4.7 Conclusions 4-14

Page

V. Feature Selection 5-1

5.1 Introduction 5-1

5.2 Feature Saliency 5-1

5.2.1 Saliency 5-1

5.2.2 Selection 5-4

5.3 Hidden Node Pruning 5-7

5.4 Results 5-7

5.5 Conclusions 5-8

VI. Theoretical Framework 6-1

6.1 Local Minimization 6-1

6.1.1 Placement 6-1

6.1.2 Fitness 6-2

6.1.3 Fusion 6-2

6.2 Global Minimization 6-2

6.3 Conclusions 6-3

VII. Conclusions and Contributions 7-1

7.1 Conclusions 7-1

7.2 Contributions 7-1

7.3 Follow on Work 7-2

Bibliography BIB-1

Vita VITA-1

List of Figures

Figure Page

2.1. The principle processing blocks are displayed for SL-MICA. Subsequent to the TrainOLNs block, the outcomes are tested for accuracy. Afterwards either more HLNs are added or RelaxHLNs block occurs. RelaxHLNs removes HLNs until the Test set accuracy declines. . . . 2-2

2.2. Hyperplane reuse is demonstrated when the base class remains the same. 2-6

2.3. A snapshot of SL-MICA placing a hyperplane is presented. 2-8

2.4. A series of snapshots is presented as the adaptive neighborhood size algorithm trains. The hyperplane initially separated the neighborhoods in (a) and also after the first increase in (b), but not after the second increase in (c). Therefore the size reverts back to the last size where the hyperplane did separate in (d). 2-11

2.5. A series of plots illustrates SL-MICA as it iterates during training. Initial training adds HLNs to separate all data points in (a) which results in the classification coloring in (b). Then SL-MICA only trains on the missed data points in (c), which results in a new classification coloring in (d). After one more iteration of training in (e) it results in 100% classification accuracy in (f). 2-15

2.6. Baum's technique for isolating two points by using three hyperplanes if the rest of the data are in general position 2-16

2.7. TrainHiddenNodes algorithm generates the HLNs in (a) and resulting classification coloring in (b). Then, the same data are trained with TrainHiddenBaum which generates the hyperplanes in (c), which results in a new classification coloring in (d). 2-19

2.8. The complete x-y coordinate grid is fed into four different MLP networks trained with the complete data set using (a) SL-MICA, (b) Backprop with momentum, (c) Backprop/HLN, which seeds Backprop with the HLN weights from SL-MICA, and (d) Backprop/OLN which seeds Backprop with all the weights from SL-MICA 2-24

2.9. Training error for the three Backprop algorithms. 2-25

Figure Page

2.10. The complete x-y coordinate grid is fed into three different MLP networks trained using SL-MICA, SL-MICA, and Backprop. These results are overlayed with the training data in (a), (c) and (e) and with the test data in (b), (d), and (f) 2-27

2.11. The classification accuracies increase with additional HLNs for the training set but, decrease for the test set. 2-30

3.1. Given data that is linearly separable and adding a dimension implies that the data is linearly separable in the new space. Moreover, the equation of the separating hyperplane remains the same. 3-3

3.2. (a)is an example of linearly separable data. (b) is the linearly separable data projected into the hidden node output space. A new point is added in (c) with two hyperplanes to separate the new data point. (d) shows the new hidden node output space with the two new hyperplanes. (e) shows the new data point on a unique vertex and how the hyperplane can be pivoted to separate the new data point as well as the original data points. 3-6

3.3. The two curves represent the Bayes bounding of the classification error of the ATM Access Control data set. 3-11

3.4. The classification accuracies are plotted of the ATM Access Control data set as a function of adding additional HLNs. 3-12

4.1. The color maps indicate the class for each pixel in the output space (a) is for a one hidden layer network and (b), is for a two hidden layer network. 4-5

4.2. The color maps indicate the class for each pixel in the output space, where (a) is for a one hidden layer network, (b), is for a two hidden layer network and (c), is for a three hidden layer network. 4-8

4.3. The classification accuracies increase for the training set but decrease for the test set. 4-10

Figure Page

4.4. The initial training of MICA results in the classification coloring in (a). Then MICA experiments with adding nodes to the same layer in (b), and adding a new layer of nodes in (c). MICA chooses to add the nodes to the same layer and then repeats the experiments in (d) and (e). Again, adding to the same layer is deemed the best 4-13

5.1. The direction of maximum separation is in the direction of the Normal to the decision boundary where the line connecting the two points cross the decision boundary. 5-2

List of Tables

Table Page

2.1. Average Spiral Data Set Results, 25 Runs 2-23

2.2. Average Mesh Data Set Results, 25 Runs 2-25

2.3. Average Iris Data Set Results, 25 Runs 2-26

2.4. Average OCR Data Set Results, 10 Runs 2-28

2.5. Average XOR Data Set Results, 25 Runs 2-29

2.6. ATM Access Control Data Set Results 2-29

2.7. ATM Access Control Data Set Results, 1 Iteration, 2 Runs 2-31

4.1. Iris Data Set Results, 25 Runs 4-4

4.2. Mesh Data Set Results, 25 Runs 4-6

4.3. OCR Data Set Results, 10 Runs 4-6

4.4. XOR Data Set Results 4-7

4.5. ATM Data Set Results, 1 Run 4-9

4.6. MICA Test Results 4-14

5.1. Iris Data Set Results, 25 Runs 5-8

5.2. OCR Data Set Results, 5 Runs 5-8

AFIT/DS/ENG/97-01

Abstract

The construction of Multi Layer Perceptron (MLP) neural networks for classification is explored. A novel algorithm is developed, the MLP Iterative Construction Algorithm (MICA), that designs the network architecture as it trains the weights of the hidden layer nodes. The architecture can be optimized on training set classification accuracy, whereby it always achieves 100% classification accuracies, or it can be optimized for generalization. The test results for MICA compare favorably with those of backpropagation on some data sets and far surpasses backpropagation on others while requiring less FLOPS to train. Feature selection is enhanced by MICA because it affords the opportunity to select a different set of features to separate each pair of classes. The particular saliency metric explored is based on the effective decision boundary analysis, but it is implemented without having to search for the decision boundaries, making it efficient to implement. The same saliency metric is adapted for pruning hidden layer nodes to optimize performance. The feature selection and hidden node pruning techniques are shown to decrease the number of weights in the network architecture from one half to two thirds while maintaining classification accuracy.

Dissertation:

Autonomous Construction of Multi Layer Perceptron Neural Networks

I. Introduction

1.1 Historical Background

Neural Networks were a budding technology in the 1950's and 1960's. Rosenblatt's neuron model, the perceptron, provided a fundamental building block for Artificial Neural Networks (ANNs) [24]. Minsky, taking into account only a small portion of Rosenblatt's research, presented a proof demonstrating that a single layer of perceptrons could not solve a problem as simple as the XOR problem [19]. This precipitated a loss of interest in neural network research despite Rosenblatt's evidence that multi-layer networks solved the XOR problem. Werbos, in his 1974 dissertation, developed a learning algorithm for multi layer perceptrons (MLPs) by propagating the errors backward through the network [36]. This work went virtually unnoticed until 1985 when Rumelhart reinvented the algorithm, calling it backpropagation or just backprop, and utilized it for training MLP neural networks [29].

MLPs trained with backprop seemed magical. They could learn weights to estimate functions and classify data, but alas, there is no magic. Neural networks have been thoroughly analyzed and their mechanisms are now understood [5,23,26]. What has not been discerned is the best network architecture for a given set of data. The variables include the number of layers, the quantity of hidden nodes in each layer, and the best features to use. Previous attempts to solve the network architecture problem resulted in rules of thumb. For example, Wildrow says the number of free weights should not exceed the number of training samples divided by ten [37]. The problem with this rule, and others like them, is that it defines an architecture that the data can support, not the best architecture to separate the data. Presently, MLP architectures are primarily determined by the rules of thumb and trial and error. A more methodical approach using statistical regression was done by Steppe [32,33]. Also, feature selection techniques are generally applied to an entire problem. This approach handicapped features that were only effective in helping to separate a subset of the problem. This dissertation defines an algorithm for constructing MLP networks which resolves the questions of network architecture and feature selection.

1.2 Problem Statement and Scope

The goal of this research is to determine how to construct a MLP artificial neural network layer by layer starting with the first layer of hidden nodes, dynamically select features depending on which classes are being separated, and prune the hidden nodes to optimize performance. This research determines the number of hidden layers needed; generates the number of hidden layer nodes needed for each layer; learns the weights for each node; selects the most advantageous features to separate each pair of classes; prunes hidden layer nodes for best generalization; proves the ability of the construction algorithm to correctly classify the data; proves the quality of the constructed solution; and proves, in general, that a framework exists whereby local learning equates to global learning.

Construction algorithm. Neural networks can be constructed by deciding where to place each hyperplane then learning their weights. The construction algorithm incorporates localized learning to train these weights. The hyperplanes are placed to optimized the separation of the data in the different classes. Hyperplanes are then placed in the hidden node output space to isolate each class. A novel algorithm, the MLP Iterative Construction Algorithm (MICA), that accomplishes the above with a single hidden layer is introduced in Chapter II. Chapter IV introduces a Multi Hidden Layer algorithm, and an algorithm that determines if adding more hidden layer nodes or adding multiple layers of nodes is best for classification accuracy and generalization.

Theoretical Basis. Backprop trained MLPs have a rich theoretical basis, but they are not guaranteed to always converge to a 100% classification accuracy on the training data. However, MICA can make this claim and it is proven in Chapter III for the Single Hidden Layer algorithm and in Chapter IV for the Multi Hidden Layer algorithm. The same chapters prove that MICA leads to a mean squared error approximation of the Bayes optimal discriminant function.

Feature Selection. There exist numerous feature saliency metrics and feature selection approaches [16,17,21,26,27,34]. The classifier based approaches generally rate features based on their proficiency in aiding the separation of the individual classes. This approach may unfairly rate a feature that performs well separating some but not all classes. The construction algorithm above adds hyperplanes to separate by class pair and thus can also rate a feature's ability to separate by class pair. This allows a feature that is superior at separating two classes, but inferior in separating the rest, to still be used without adversely affecting the classification accuracies for the remainder of the classes. This unique feature selection approach is demonstrated with one saliency metric in Chapter V.

Hidden Node Pruning. Once a network is trained or constructed, there is a set architecture that is based on the training set. The training set may have idiosyncrasies not found in the data as a whole. Nevertheless, these idiosyncrasies have influenced the weights and the architecture found by the construction algorithm. However, the construction algorithm offers a unique metric for determining the importance of each hidden node in the network. Thus, some may be removed to discount the idiosyncrasies and increase generalization. Chapter II offers a method to prune hidden nodes based on the importance metric to maximize generalization. The construction approach may create redundant hidden nodes or superfluous hidden nodes with respect to training a single output layer node. Chapter V describes an innovative approach to reduce hidden layer nodes while training output layer weights.

Theoretical Framework. The unique construction approach used herein is predicated upon minimizing a set of local errors to minimize the global error. Yet, the results indicate this approach not only equals, but often surpasses global minimization attempts. An analysis to determine what conditions and assumptions are necessary for local minimization to equal or surpass global minimization is located in Chapter VI. This provides the general framework which defines the necessary conditions for local optimization to equate to global optimization.

1.3 Dissertation Organization

Chapter II develops the Single Hidden Layer MLP Iterative Construction Algorithm and produces the basic algorithmic blocks used in the following chapters. Chapter III proves the Single Hidden Layer MLP Iterative Construction Algorithm can produce an MLP to correctly classify any training set in general position to 100% accuracy. A separate proof affirms that the algorithm is Bayes optimal. Chapter IV introduces the Multiple Hidden Layer MLP Iterative Construction Algorithm and its theoretical basis. A unified construction algorithm that decides the best network architecture concludes the chapter. Feature Selection and Hidden Node Pruning are discussed in Chapter V. The theoretical framework is mapped out in Chapter VI. Chapter VII contains conclusions and contributions.

II. Constructing Single Hidden Layer MLPs

2.1 Introduction

This chapter details a novel algorithm for constructing single hidden layer Multi Layer Perceptron (MLP) neural networks for use in classification problems. The Single Layer MLP Iterative Construction Algorithm (SL-MICA) realizes a Cybenko-like net, i.e., one with a single hidden layer. Cybenko [7] proved that given enough nodes, a single hidden layer MLP can classify any training data such that the total measure of the incorrectly classified points can be made arbitrarily small. Cybenko, in his closing remarks states, "The important questions that remain to be answered deal with feasibility, namely how many terms in the summation (or equivalently, how many neural nodes) are required to yield an approximation of a given quality [7]." Many researchers have tried to answer Cybenko's question simply from the characteristics of the data [1–3,6,30,31,35], but much of this research is impractical. For example, Baum's Theorem 1 states, "A one-hidden-layer net with $\lceil N/d \rceil$ internal units can compute an arbitrary dichotomy of N d-dimensional vectors in general position [1]." The problem is that one usually wants to compute a specific dichotomy not an arbitrary one, and thus $\lceil N/d \rceil$ hidden nodes may be too many and adversely effect generalization. Suppose one wishes to solve the two dimensional XOR problem with a 1000 data points. Baum's theorem suggests that we need 500 hidden nodes, where, in reality, four may suffice. Another well known work is Vapnik's VC dimension or capacity [35], which defines the maximum number of data points for which a classifier can be implemented on all possible dichotomies. Baum and Haussler have derived bounds on the VC dimension in terms of the number of hidden nodes, total nodes, and total number of weights [3]. However, these bounds are very conservative and led Jain and Mao to state, "The practical applicability of these bounds is questionable [14]." It is apparent that the number of hidden nodes needed is data dependent and cannot be gleaned except by processing the data. It is impractical to use backprop, and similar techniques, to design network architectures, because they can only minimize a global error. This error would be unknown if the network architecture is not already established and the weights trained. Thus, one is left with trial and error if one wants to use Backprop for learning and network selection. Therefore, a construction approach, where the net is built layer by layer, is the logical choice for architecture selection since it will take the data into account when constructing the network. A theory on this subject is presented in a paper by Roy [25]. Roy proffers robust and efficient learning theory and states that an intelligent learning algorithm should have the following general characteristics:

1. Perform network design task
2. Robustness in learning
3. Quickness in learning
4. Efficiency in learning
5. Generalization in learning

Construction algorithms [4,39] are becoming more prevalent. SL-MICA not only embodies the above characteristics, but is unique in that it designs a standard MLP network. Moreover, SL-MICA can construct a network where the number of missed points is made arbitrarily small. SL-MICA does not reduce the global error, but does minimize local errors. This makes SL-MICA an exceptionally powerful construction technique. The remainder of this chapter details the construction techniques of SL-MICA and presents comparisons to Backprop.

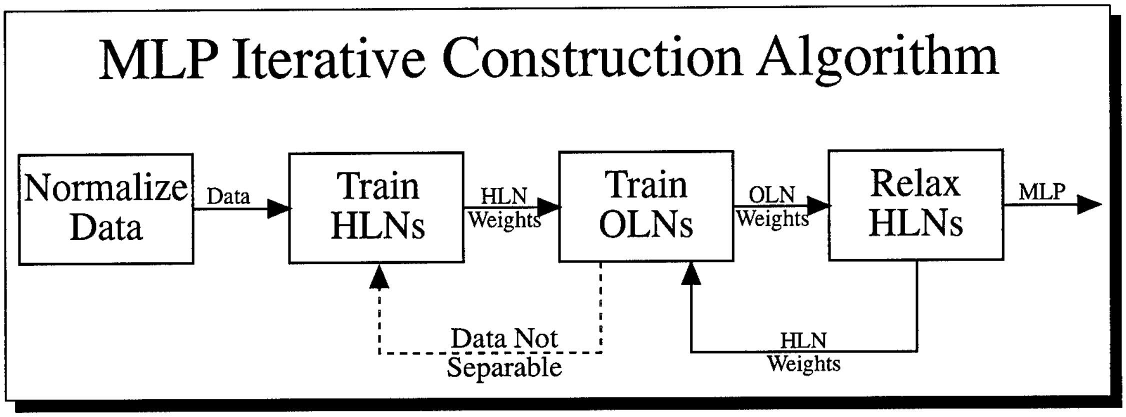


Figure 2.1 The principle processing blocks are displayed for SL-MICA. Subsequent to the TrainOLNs block, the outcomes are tested for accuracy. Afterwards either more HLNs are added or RelaxHLNs block occurs. RelaxHLNs removes HLNs until the Test set accuracy declines.

A high level block diagram of SL-MICA is presented in Figure 2.1. The normalization routine simply scales each dimension of the data to a zero mean and standard deviation of one, while also augmenting the data matrix with a column of ones. The Ho-Kashyap algorithm is central to training both the hidden layer nodes (TrainHLNs) and output layer nodes (TrainOLNs). A brief overview is provided in Section 2.2. The TrainHLNs and TrainOLNs blocks are discussed in Sections 2.3 and 2.4 respectively. The iteration training of SL-MICA is analyzed in Section 2.5. The relaxation block increases the generalization ability of SL-MICA and is explained in Section 2.6. Test results are presented in Section 2.7 followed by concluding remarks in Section 2.8.

2.2 Ho-Kashyap Method

SL-MICA is a hybrid process that exercises both hyperplane placement and error minimization to design a MLP. SL-MICA selects an area in the feature space to place a hyperplane, then minimizes a local error to maximize the hyperplane's effectiveness for separation. The hyperplanes are placed by selecting two sets of opposite class data to separate. A hyperplane is trained to fit amidst the data sets by minimizing the mean squared error of a discriminant vector. The Ho-Kashyap method fulfills this assignment.

There are alternative methods for finding a hyperplane to separate linearly separable data sets; however, the Ho-Kashyap method has several desirable qualities. Ho-Kashyap's method is guaranteed to converge in a finite number of iterations if the data are linearly separable [8, pp 161-163], and the method has an indicator if the data are not linearly separable. The Ho-Kashyap method is a gradient descent algorithm, whereby it minimizes $\left\|\mathbf{X}\vec{w} - \vec{t}\,\right\|$, where $\mathbf{X}$ is a matrix containing the data and $\vec{w}$ and $\vec{t}$ are to be determined. If the data are linearly separable, then there exists a $\vec{w}_o$ and $\vec{t}_o$ such that

$$\begin{aligned} \mathbf{X}\,\vec{w}_o &= \vec{t}_o \\ \vec{t}_o &> 0 \end{aligned} \tag{2.1}$$

$\mathbf{X}$ is a matrix of feature or data vectors augmented with the value one. It is shown below for a two-class problem with n data vectors in class 1 and m data vectors in class 2. The data in the second class are multiplied by -1 so that the least squares solution to $\vec{w}$, called the discriminant vector, will attempt to place it between the two data sets. The vector $\vec{t}$ must contain all positive elements for this process to succeed.

$$\mathbf{X} = \begin{bmatrix} x^{1}{}_{1,1} & x^{1}{}_{1,2} & \cdots & x^{1}{}_{1,k} & 1 \\ x^{1}{}_{2,1} & x^{1}{}_{2,2} & \cdots & x^{1}{}_{2,k} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x^{1}{}_{n,1} & x^{1}{}_{n,2} & \cdots & x^{1}{}_{n,k} & 1 \\ -x^{2}{}_{1,1} & -x^{2}{}_{1,2} & \cdots & -x^{2}{}_{1,k} & -1 \\ -x^{2}{}_{2,1} & -x^{2}{}_{2,2} & \cdots & -x^{2}{}_{2,k} & -1 \\ \vdots & \vdots & & \vdots & \vdots \\ -x^{2}{}_{m,1} & -x^{2}{}_{m,2} & \cdots & -x^{2}{}_{m,k} & -1 \end{bmatrix}$$

The superscript indicates class. The error is $\vec{e} = \mathbf{X}\vec{w} - \vec{t}$. If $\vec{e}$ contains all positive values, then the data sets are separated by the hyperplane defined by $\vec{w}$. If $\vec{e}$ contains negative values, then the elements in $\vec{t}$ that correspond to the elements in $\vec{e}$ with positive values are increased proportionally to their error and $\vec{w}$ is then recalculated. This has the effect of moving $\vec{w}$ away from data on the wrong side of the discriminant. If all values in $\vec{e}$ are negative then the data is not linearly separable. The Ho-Kashyap method as presently described requires $\vec{w}$ to be recalculated for each iteration from the pseudo-inverse and $\vec{t}$. However, the equations can be rewritten with an incremental update of $\vec{w}$ using $\vec{e}$. This leads to Algorithm 1.

Algorithm 1 Ho-Kashyap Method

```
$\vec{t}_1 \leftarrow 1$ {This initial value is arbitrary but must be greater than 0}
$\vec{w}_1 \leftarrow \mathbf{X}^\dagger \vec{t}_1$ {X is training data, setup according to the problem, † defined below}
$\vec{e}_1 \leftarrow \mathbf{X}\vec{w}_1 - \vec{t}_1$
$i \leftarrow 1$
while $\vec{e}_i \neq \epsilon$ do
  $\vec{t}_{i+1} \leftarrow \vec{t}_i + \rho(\vec{e}_i + |\vec{e}_i|)$ {$\rho$ is an update parameter, $0 < \rho < 1$}
  $\vec{w}_{i+1} \leftarrow \vec{w}_i + \rho\mathbf{X}^\dagger |\vec{e}_i|$ {$\mathbf{X}^\dagger = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$ is the left pseudo-inverse}
  $\vec{e}_i \leftarrow \mathbf{X}\vec{w}_i - \vec{t}_i$
  $i \leftarrow i + 1$
end while
```

The $\mathbf{X}$, $\mathbf{X}^\dagger$, *and* ϵ values are provided as inputs. The Ho-Kashyap algorithm calls for ϵ to be close to zero, but larger values can be used. For a complete derivation of the algorithm and proof of convergence if given linearly separable data see [8, pp 159-166].

2.3 Training Hidden Layer Nodes

The TrainHLN block of SL-MICA trains a block of Hidden Layer Nodes (HLNs) such that all the data in each class are separated from all data in every other class. The term separated is used to mean a hyperplane of the "correct orientation" exists between the data. "Correct orientation" is explained below. This is a necessary condition for making the data linearly separable when they are projected into the hidden node space. Chapter III details the particulars of linear separability. TrainHLN lacks additional stopping criteria, so it invariably trains enough HLNs to separate all the data in a class from all the data in the other classes. However, a metric of importance is determined for each HLN produced so that eventually they may be pruned. This is discussed in Section 2.6.

MICA trains HLNs in a four step process. First, SL-MICA selects a pair of classes to separate. Second, SL-MICA finds the inter-class data pair with the smallest Euclidean distance that has not yet been separated by a hyperplane. Third, SL-MICA searches for the largest possible group of neighboring points around the inter-class data pair that can be separated by a single hyperplane. Fourth, the remaining unseparated inter-class data pairs are tested to see if the new hyperplane separates them. Two classes are separated when every inter-class data pair is separated. If there are unseparated data pairs, the algorithm reverts to step 2, otherwise it returns to step 1. Once all classes are separated, the algorithm quits, see Algorithm 2.

Algorithm 2 TrainHiddenNodes

```
while more class pairs $i$ and $j$ do
  $\mathbf{Class}_i \leftarrow \mathbf{ClassData}$ {Step 1, Transfer all data pertaining to class $i$}
  $\mathbf{Class}_j \leftarrow \mathbf{ClassData}$ {Step 1, Transfer all data pertaining to class $j$}
  $\mathbf{D} \leftarrow distance(\mathbf{Class}_i, \mathbf{Class}_j)$ {Step 2, Equation 2.3}
  while $\mathbf{D}_{l,m} \neq \infty \forall l, m$ do
    $[row, column] \leftarrow min(\mathbf{D})$ {Step 2}
    $\mathbf{X}_1 \leftarrow \mathbf{Class}_i(row, :)$ {Step 2, Transfer the data vector}
    $\mathbf{X}_2 \leftarrow \mathbf{Class}_j(column, :)$ {Step 2, Transfer the data vector}
    $\mathbf{Weight}_k \leftarrow AdaptNeighborhood(\mathbf{Class}_i, \mathbf{Class}_j, \mathbf{X}_1, \mathbf{X}_2)$ {Step 3}
    $\mathbf{D} \leftarrow TestForSeparation(\mathbf{D}, \mathbf{Weight}_k)$ {Step 4}
  end while
end while
```

2.3.1 Selecting Class Pairs. MICA separates classes on a pairwise basis. For example, in a three-class problem, data from classes one and two are separated, then the data from classes one and three, and finally data from classes two and three. Separating all classes on a pair-wise basis is effective, but possibly inefficient because of the need for redundant nodes. Figure 2.2 illustrates how an HLN that separates classes one and two can separate classes one and three. Thus, MICA has an option of reusing HLNs to limit the creation of redundant HLN.

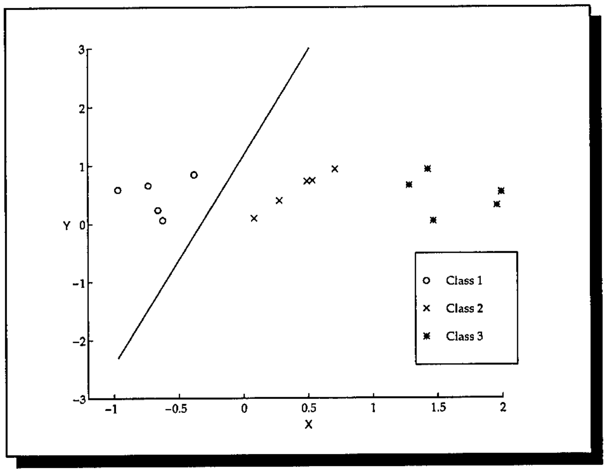


Figure 2.2 Hyperplane reuse is demonstrated when the base class remains the same.

Reuse can be employed whenever the base class remains the same. The base class is the class with the smaller value, and in the previous example would be class one. In a three class problem, there is only one chance of reusing HLNs, but in a four class problem this expands to three and in a five class problem there are six chances. Equation 2.2 calculates this value,

$$\mathit{ReuseChances} = \sum_{i=1}^{N-1} i - 1 \tag{2.2}$$

where N is the number of classes. In larger class problems the number of reuse chances becomes significant and leads to large reductions in the number of HLNs needed for separating the data. Each time the base class remains the same, the number of HLNs involved in reuse increases. If SL-MICA is separating classes one and three, it is only reusing HLNs that separated classes one and two. Whereas, if SL-MICA is separating classes one and four, it is reusing HLNs that separated classes one and two, and HLNs that separated classes one and three. However, when SL-MICA separates classes two and four, it is only reusing HLNs that separated classes two and three. This is reasonable since SL-MICA adopts the convention that the base class is always on the high side of the hyperplane. This allows SL-MICA to more easily keep track of the orientation specific nature of the hyperplanes.

2.3.2 Selecting Inter-class Data Pairs. After the pair of classes are selected for separation, a distance matrix is calculated that contains a distance value between every pair of points in the two classes. SL-MICA selects the inter-class data pair that are the closest together using Euclidean distance. This distance matrix is used to select the nearest neighbor inter-class data pairs and keeps track of which pairs have already been been separated. Let **D** denote the distance matrix whose entries are the distances for each inter-class data pair.

$$\mathbf{D} = \begin{bmatrix} \|{p^1}_1 - {p^2}_1\| & \|{p^1}_1 - {p^2}_2\| & \cdots & \|{p^1}_1 - {p^2}_m\| \\ \\ \|{p^1}_2 - {p^2}_1\| & \|{p^1}_2 - {p^2}_2\| & \cdots & \|{p^1}_2 - {p^2}_m\| \\ \vdots & \vdots & & \vdots \\ \|{p^1}_n - {p^2}_1\| & \|{p^1}_n - {p^2}_2\| & \cdots & \|{p^1}_n - {p^2}_m\| \end{bmatrix} \tag{2.3}$$

where ${p^i}_k$ is the ith data vector from the kth class. The indices of the smallest distance value in the matrix are used to select the nearest neighbor inter-class data pair. Let (i, j) be the location of the smallest value in **D**. The corresponding points are derived from the class data as follows: ${\mathbf{p}^1}_i = {Class^1}_i$ *and* ${\mathbf{p}^2}_j = {Class^2}_j$. A distance value of zero indicates an inter-class data pair is not in general position. Each time a selection is made it is replaced with a large value equal to the computation infinity. SL-MICA adopts the convention that a symbol of ∞ in **D** indicates that an inter-class data pair is already separated. Thus, two classes are completely separated if $\mathbf{D}_{i,j} = \infty, \forall$ i, j.

2.3.3 Selecting Class Neighborhoods. To achieve the best generalization, neighborhood selection is a crucial aspect of SL-MICA. A neighborhood is a set of homogeneous class vectors. The larger the neighborhood, the better the generalization and the faster the algorithm converges. Two neighborhoods are selected, one for each class being separated. Figure 2.3 is a snapshot of SL-MICA training. This snapshot is taken as SL-MICA is placing the first HLN. The hyperplane must separate the inter-class data pair pointed to by the arrow. The $\otimes$'s are the class 1 data in the neighborhood of the inter-class data pair member from class 1 and the $\oplus$'s are the class 2 data in the neighborhood of the inter-class data pair member from class 2. The hyperplane is trained using the Ho-Kashyap algorithm, see algorithm 1, and utilizes the neighborhood data as input.

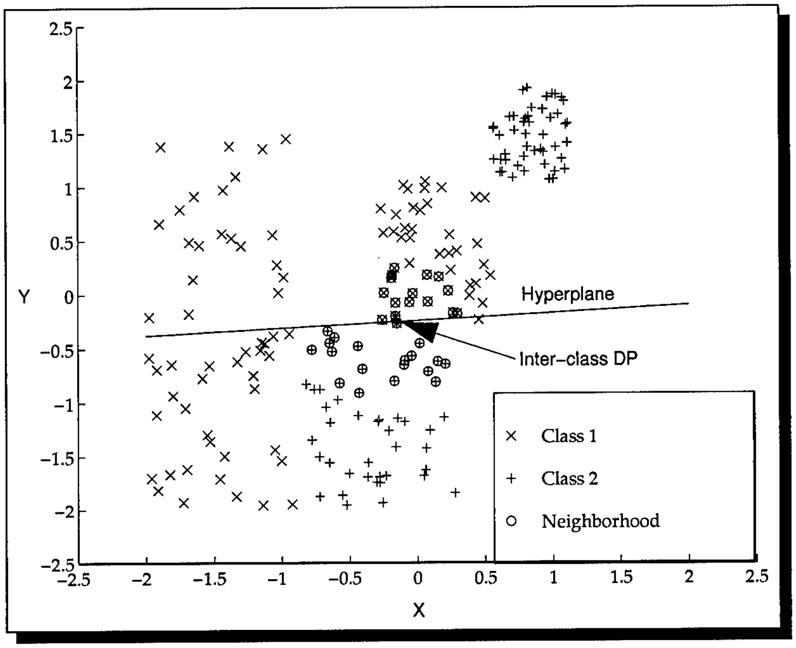


Figure 2.3 A snapshot of SL-MICA placing a hyperplane is presented.

Neighbors are selected by first calculating the distances between the inter-class data pair members and the data from their respective classes. The data whose distances are less than x number of standard deviations away are designated in the neighborhood. Larger neighborhoods are preferred since, in general, they require fewer HLNs and result in less complex decision boundaries. However, if the neighborhood size is too large, the data may not be linearly separable. Therefore, an adaptable neighborhood size algorithm is used; see Algorithm 3.

Algorithm 3 AdaptNeighborhoood

```
$xstd \leftarrow x$
repeat
    $xstd \leftarrow xstd + \Delta std$
    $\mathbf{Neighborhood}_1 \leftarrow GetNeighbors\,(Class_1, x_1, xstd)$
    $\mathbf{Neighborhood}_2 \leftarrow GetNeighbors\,(Class_2, x_2, xstd)$
    $\mathbf{X} \leftarrow [\mathbf{Neighborhood}_1; -\mathbf{Neighborhood}_2]$ {This places the negative of the neighborhood 2 vectors under the neighborhood 1 vectors}
    $\mathbf{X}^\dagger \leftarrow (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$
    $weights \leftarrow HoKashyap\left(\mathbf{X}, \mathbf{X}^\dagger, \epsilon\right)$ {Call to Ho-Kashyap Algorithm 1}
until $\neg separated\,(\mathbf{Weights}, \mathbf{Neighborhood}_1, \mathbf{Neighborhood}_2)$
repeat
    $xstd \leftarrow xstd - \Delta std$
    $\mathbf{Neighborhood}_1 \leftarrow GetNeighbors\,(\mathbf{Class}_1, x_1, xstd)$
    $\mathbf{Neighborhood}_2 \leftarrow GetNeighbors\,(\mathbf{Class}_2, x_2, xstd)$
    $\mathbf{X} \leftarrow [\mathbf{Neighborhood}_1; -\mathbf{Neighborhood}_2]$ {This places the negative of the $\mathbf{Neighborhood}_2$ vectors under the $\mathbf{Neighborhood}_1$ vectors}
    $\mathbf{X}^\dagger \leftarrow (\mathbf{X}^t\mathbf{X})^{-\ 1}\mathbf{X}^t$
    $\mathbf{Weights} \leftarrow HoKashyap\left(\mathbf{X}, \mathbf{X}^\dagger, \epsilon\right)$ {Call to Ho-Kashyap Algorithm 1}
until $separated\,(weight, \mathbf{Neighborhood}_1, \mathbf{Neighborhood}_2)$
```

This algorithm searches to find the largest possible neighborhood size, in terms of standard deviations, for a given inter-class data pair. The neighborhood size is initialized to some value and if all opposite class vectors in the neighborhoods are separated by the resulting hyperplane, the neighborhood size grows until the neighborhoods are no longer separable. The neighborhood size is then reduced to the previous level where the neighborhoods were separable. If the initial neighborhood size is too large, the neighborhood size is stepped down until the neighborhoods become separable. SL-MICA is able to quickly adapt the neighborhood size since it can ascertain whether the neighborhoods are separated by the hyperplane from inspection of the error vector from the Ho-Kashyap algorithm. Otherwise, considerably more processing time is required to perform separation testing on all possible pair combinations from the two neighborhoods. The call to the Ho-Kashyap method, see Algorithm 1, is done with a large ϵ value for efficiency reasons. The downside is that the neighborhood sizes may be smaller than would otherwise be the case, but the increase in efficiency offsets this drawback.

Figure 2.4 shows a series of snapshots as the neighborhood size adapts. In Figure 2.4(a) the hyperplane separates the neighborhoods from the initial neighborhood size. In Figure 2.4(b) the neighborhood size is larger but the hyperplane still separates the neighborhoods. In Figure 2.4(c) the size is too large and the hyperplane cannot separate the neighborhoods. In Figure 2.4(d), the neighborhood size is stepped back down and thus, they become separable again.

Also key to generalization is the positioning of the hyperplanes. They should be roughly halfway between the two data points in inter-class data pair. If the above neighborhood selection approach fails to accomplish this goal, then the neighborhood size defaults to a radius around each member of the inter-class data pair that is slightly less than the distance between the two data points. If this approach does not result in a good hyperplane, then as a fail safe, the vector normal to the line connecting the two inter-class data points $\mathbf{P}_1, \mathbf{P}_2$ are used to define the separating hyperplane, see Eq 2.4,

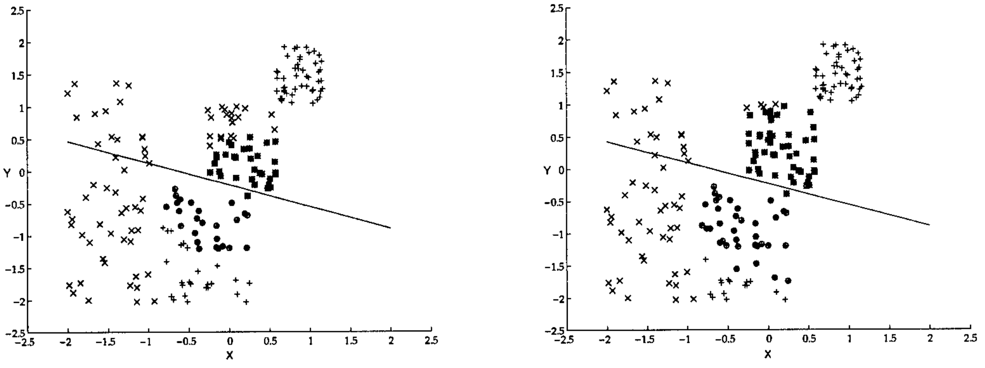
$$(\mathbf{P}_1 - \mathbf{P}_2) \cdot (X - \mathbf{P}_m) = 0 \tag{2.4}$$

where $\mathbf{P}_m = \frac{(\mathbf{P}_1 + \mathbf{P}_2)}{2}$ is the midpoint between the inter-class data points $\mathbf{P}_1$ and $\mathbf{P}_2$.

2.3.4 Testing for Separability. A hyperbolic tangent activation function makes determining separability straight forward. SL-MICA tests for opposite polarity of the output of the HLN, using the following equation:

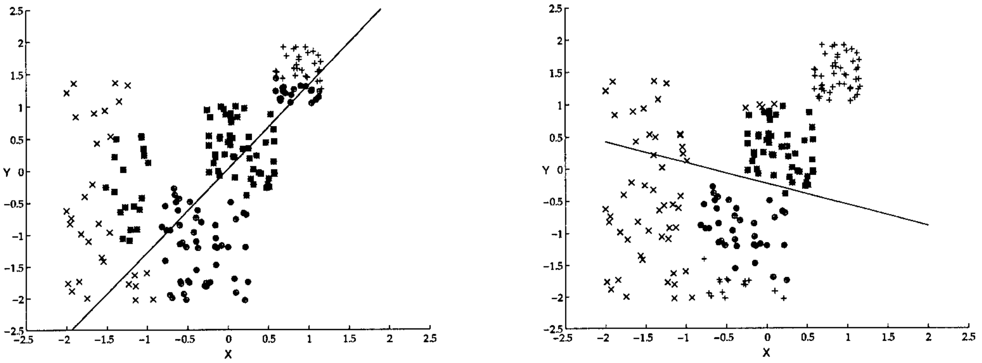
$$\left|y_j^1 + y_j^2\right| = \left|y_j^1\right| + \left|y_j^2\right| \tag{2.5}$$

where the superscript indicates class and y_j is the output from the jth HLN. If the equality in Eq. 2.5 is true, then the points are not separated by the hyperplane. Once a hyperplane has separated two points it becomes “orientation specific.” If a hyperplane separates class 1 to the positive side and class 2 to the negative side, then that hyperplane can only separate other inter-class data pairs if they fall on the correct side of the hyperplane. SL-MICA forces the orientation of the hyperplane such that the base class is always on the positive side of the hyperplane. The base class is the class with the smaller index number. Thus,



(a) Initial Neighborhood size, Hyperplane Separates

(b) Neighborhood size increases, Hyperplane Separates



(c) Neighborhood size increases, Hyperplane does not Separates

(d) Neighborhood size decreases, Hyperplane Separates

Figure 2.4 A series of snapshots is presented as the adaptive neighborhood size algorithm trains. The hyperplane initially separated the neighborhoods in (a) and also after the first increase in (b), but not after the second increase in (c). Therefore the size reverts back to the last size where the hyperplane did separate in (d).

class 1 data are always on the positive side of the hyperplanes from classes 2 and 3 data, and class 2 data are always on the positive side of the hyperplanes from class 3 data.

The process of searching from the minimum value in the **D** matrix to find the next inter-class data pair to separate is inefficient, because the inter-class data pair that has the smallest distance may already be separated. Therefore, each time an HLN is produced, each inter-class data pair in the **D** matrix is tested to see if the new HLN separated it. Inter-class data pairs already marked "separated" are skipped each successive round and processing is expedited for each new HLN produced. There is one shortcut that is employed to bypass some of the testing of the inter-class data pairs in the **D** matrix. When a new HLN is produced, it must separate all opposite class data in the two neighborhoods. Therefore, all possible inter-class data pair combinations from the two neighborhoods can be marked separated without having to test them for verification. Thus, the **D** matrix can be altered in the process of finding the nearest neighbor inter-class data pairs, the process of finding the HLNs, and the process of testing for separability.

2.4 Training Output Layer Weights

The TrainHLN block produces a set of HLNs with the necessary conditions for linearly separable data. To finish the training, the TrainOLN block places hyperplane between the data in one class and the data in all other classes. The **X** matrix contains the data in class i and the negative of all other data not in class i as follows,

$$\mathbf{X} = \begin{bmatrix} {x^{i}}_{1,1} & {x^{i}}_{1,2} & \cdots & {x^{i}}_{1,k} & 1 \\ {x^{i}}_{2,1} & {x^{i}}_{2,2} & \cdots & {x^{1}}_{2,k} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ {x^{i}}_{n,1} & {x^{i}}_{n,2} & \cdots & {x^{i}}_{n,k} & 1 \\ -{x^{\neg i}}_{1,1} & -{x^{\neg i}}_{1,2} & \cdots & -{x^{\neg i}}_{1,k} & -1 \\ -{x^{\neg i}}_{2,1} & -{x^{\neg i}}_{2,2} & \cdots & -{x^{\neg i}}_{2,k} & -1 \\ \vdots & \vdots & & \vdots & \vdots \\ -{x^{\neg i}}_{m,1} & -{x^{\neg i}}_{m,2} & \cdots & -{x^{\neg i}}_{m,k} & -1 \end{bmatrix}$$

If the data are linearly separable the Ho-Kashyap method will produce a discriminant to separate the data in class i. SL-MICA trains one OLN per class. The Ho-Kashyap algorithm iterates until the error equals zero. When a solution is found for each OLN, the training is completed and the entire set of training data is guaranteed to be classified with 100 percent accuracy. The TrainOutputNodes algorithm is shown below.

Algorithm 4 TrainOutputNodes

```
$\mathbf{X} \leftarrow -[Data, \mathbf{1}^t]$ {1 is a vector of ones to augment the data}
for $i = 1 : NumClasses$ do
  $\mathbf{X_i} \leftarrow -\mathbf{X_i}${This makes the data for class i positive and the rest negative}
  $\mathbf{X}^\dagger \leftarrow (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$
  Weights$_i \leftarrow HoKashyap\left(\mathbf{X}, \mathbf{X}^\dagger, \epsilon\right)$ {Call to Ho-Kashyap Algorithm 1}
end for
```

There are two considerations for this. First, the data from TrainHLN block may not be linearly separable and secondly, the convergence with a large data set may be slow. Often a good solution can be found quickly with the error driven to within an epsilon of zero. Algorithm 5 takes advantage of this by checking classification results periodically for 100% accuracy. If the accuracy is less than 100%, then the epsilon is lowered and the training resumes using the previous $\vec{w}$ and $\vec{t}$ vectors. With ϵ initially set to 5×10^{-3} this algorithm usually converges in one iteration for the data that are linearly separable. If this is not the case, then Algorithm 5 still produces its best solution within a few iterations. Therefore, MaxIterations can be set to five or less. Section 2.5 deals with data sets that are not projected into a linearly separable space by the TrainHiddenNodes algorithm..

Algorithm 5 New TrainOutputNodes

```
$\epsilon \leftarrow 5 \times 10^{-3}$
$Accuracy \leftarrow 0$
while $Accuracy \neq 100 \wedge iterations < MaxIterations$ do
  $iterations \leftarrow interations + 1$
  $weights \leftarrow TrainOutputNodes\,(Data, \epsilon)$ {Call to TrainOutputNodes Algorithm 4}
  $Accuracy \leftarrow TestForAccuracy\,(Data, weights)$
  $\epsilon \leftarrow \epsilon/10$
end while
```

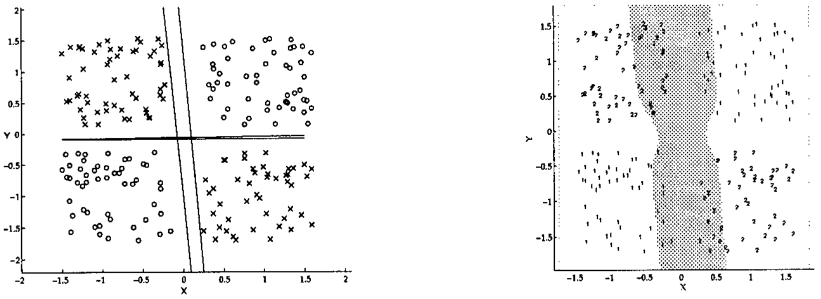
2.5 Iteration of HLN Training

If the TrainHiddenNodes algorithm does not project the data into a space where it is linearly separable, then the TrainOutputNodes algorithm can not generate hyperplanes to correctly classify all data in the training set. If 100% classification accuracy is the goal in a single hidden layer MLP network, there are two aspects to extending the above algorithm.

The first extension extracts the data points that are misclassified in the current MLP architecture and then trains more HLNs to separate those points. Next, combine the two sets of HLNs and retrain the OLNs. Figure 2.5 illustrates this approach. Figure 2.5(a) using the complete training data produces the first set of HLNs, which results in the classification coloring in Figure 2.5(b). Any points not colored correctly are added to a training data subset and new HLNs are trained to separate the data in the subsets. Figure 2.5(c) illustrates the missed training data subsets and the resulting hyperplanes to separate them. The new hyperplane set is unioned with the first hyperplane set and after retraining the OLNs results in the coloring in Figure 2.5(d). The processed is repeated in Figure 2.5(e) and results in the 100% classification accuracy shown in Figure 2.5(f). A problem arises when the newly missed training data subset is the same as the old missed training data subset. Since there is no randomness in the placement of the hyperplanes, an infinite loop results. This problem is solved in the second extension.

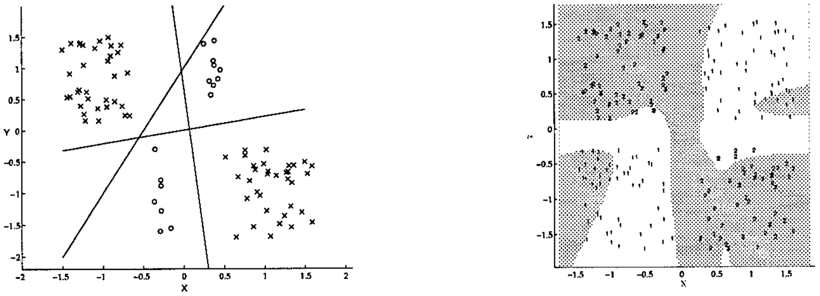
The second extension adds a set of three hyperplanes for each inter-class data pair that is unseparated. If the data are in general position, then Baum's technique [1] can isolate any two data vector pairs by using three hyperplanes. The technique calculates the weights of a hyperplane that contains the two data vectors. Then that hyperplane is offset $+\epsilon$ and a parallel duplicate is offset $-\epsilon$. Finally, a third hyperplane is drawn that is normal to both of the offset hyperplanes and intersects the midpoint of the line that connects the two data vectors. This completes Baum's technique, which is illustrated in Figure 2.6.

It is complex to calculate the weights for the hyperplanes because the equations to solve are under-determined in a linear algebraic sense. Therefore, the weights of the hyperplanes are calculated through a weight update equation. If a data vector lies in a hyperplane, then the inner product between the weights of the hyperplane and the data vector will be zero. Since a hyperplane has one more parameter than dimensions of the space, a 1 is appended to the data vector for the inner product to calculate correctly. This leads to the following error equation:



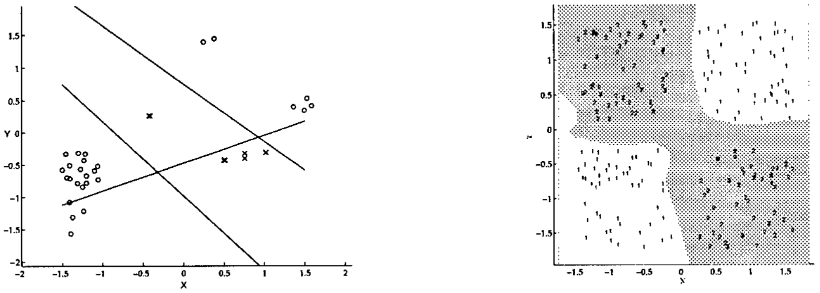
(a) Training Data and Initial Hyperplanes

(b) Classification Coloring after initial Hyperplanes



(c) Missed Data and Second Hyperplane Set

(d) Classification Coloring after Second Hyperplane Set



(e) Missed Data and Third Hyperplane Set

(f) Classification Coloring after Third Hyperplane Set

Figure 2.5 A series of plots illustrates SL-MICA as it iterates during training. Initial training adds HLNs to separate all data points in (a) which results in the classification coloring in (b). Then SL-MICA only trains on the missed data points in (c), which results in a new classification coloring in (d). After one more iteration of training in (e) it results in 100% classification accuracy in (f).

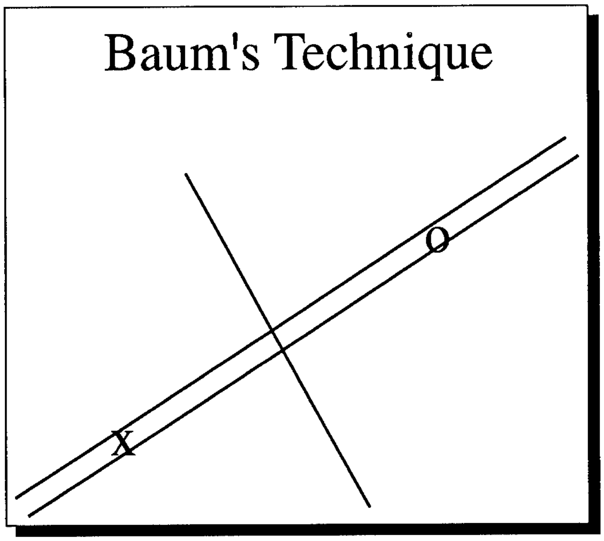


Figure 2.6 Baum's technique for isolating two points by using three hyperplanes if the rest of the data are in general position

$$e = \frac{(\mathbf{W} \cdot [\mathbf{P}_1, 1])^2}{2} + \frac{(\mathbf{W} \cdot [\mathbf{P}_2, 1])^2}{2} \tag{2.6}$$

where $\mathbf{W}$ are the weights of the hyperplane and $\mathbf{P}_i$ is a data vector in the feature space. Driving this equation to zero results in a hyperplane containing the two data vectors. This is accomplished with the following weight update equation:

$$\mathbf{W}^+ = \mathbf{W}^- + \triangle\mathbf{W} \tag{2.7}$$

where $\mathbf{W}^+$ is the updated $\mathbf{W}$, $\mathbf{W}^-$ is the current value, and $\triangle\mathbf{W}$ means the change in $\mathbf{W}$. The change in $\mathbf{W}$ is determined by taking the partial derivative of the error with respect to $\mathbf{W}$. Then stepping in the negative direction, as follows:

$$\triangle\mathbf{W} = -\eta \frac{\partial e}{\partial \mathbf{W}}, \tag{2.8}$$

where η is the step size and is a positive small number (0.001). The partial derivative of the error with respects to the weights is in Equation 2.6 is:

$$\frac{\partial e}{\partial \mathbf{W}} = (\mathbf{W} \cdot [\mathbf{P}_1, 1])\,[\mathbf{P}_1, 1] + (\mathbf{W} \cdot [\mathbf{P}_2, 1])\,[\mathbf{P}_2, 1]\,. \tag{2.9}$$

This leads to the complete weight update equation:

$$\mathbf{W}^{+} = \mathbf{W}^{-} - \eta\,(\mathbf{W}\cdot[\mathbf{P}_1,1])\,[\mathbf{P}_1,1] + (\mathbf{W}\cdot[\mathbf{P}_2,1])\,[\mathbf{P}_2,1]\,. \tag{2.10}$$

Equation 2.10 calculates a hyperplane containing the two vectors. For the Baum technique, the hyperplanes are offset to straddle the points. The weight vector has one value for each dimension of the feature space plus one more for a threshold value. Therefore, to create the sandwiching hyperplanes, positive ϵ is added to the threshold weight for the upper hyperplane while a negative ϵ is added to the threshold weight for the lower hyperplane. The normal hyperplane is calculated as in Equation 2.4. This results in the following algorithm.

Algorithm 6 TrainBaumHLNs

```
$\mathbf{W}_0 \leftarrow random$
$i \leftarrow 0$
**repeat**
  $i \leftarrow i + 1$
  $\mathbf{W}_i = \mathbf{W}_{i-1} - \eta\,(\mathbf{W}_{i-1}\cdot\mathbf{P}_1)\,\mathbf{P}_1 + (\mathbf{W}_{i-1}\cdot\mathbf{P}_2)\,\mathbf{P}_2$
**until** $\frac{(\mathbf{W}_i\cdot\mathbf{P}_1)^2}{2} + \frac{(\mathbf{W}_i\cdot\mathbf{P}_2)^2}{2} = 0$
$\mathbf{Weight}_1 \leftarrow \mathbf{W}$
$\mathbf{Weight}_{1,k} \leftarrow \mathbf{Weight}_{1,k} + \epsilon$ {In $\mathbf{Weight}_{1,k}$ the $k$ refers to the last value in the vector}
$\mathbf{Weight}_2 \leftarrow \mathbf{W}$
$\mathbf{Weight}_{2,k} \leftarrow \mathbf{Weight}_{2,k} + \epsilon$
$\mathbf{Weight}_3 \leftarrow \mathbf{P}_1 - \mathbf{P}_2$
$\mathbf{Weight}_{3,k} \leftarrow -\frac{(\mathbf{P}_1-\mathbf{P}_2)\cdot(\mathbf{P}_1+\mathbf{P}_2)}{2}$
```

The TrainBaumHLNs algorithm takes the place of the AdaptNeighborhood algorithm in the TrainHiddenNodes algorithm and TestForSeparation is removed, which results in an algorithm for training HLNs with Baum's isolation technique and is presented in Algorithm 7.

A simple experiment applying TrainHiddenBaum to the XOR data set demonstrates the effectiveness of the algorithm. Figure 2.7(a) shows the hyperplanes generated by TrainHiddenNodes and Figure 2.7(b) colors the resulting output space. Compare that to Figure 2.7(c) which uses TrainHiddenBaum to calculate the hyperplanes and the resulting output coloring in Figure 2.7(d). There are more hyperplanes in Figure 2.7(c) than can

Algorithm 7 TrainHiddenBaum

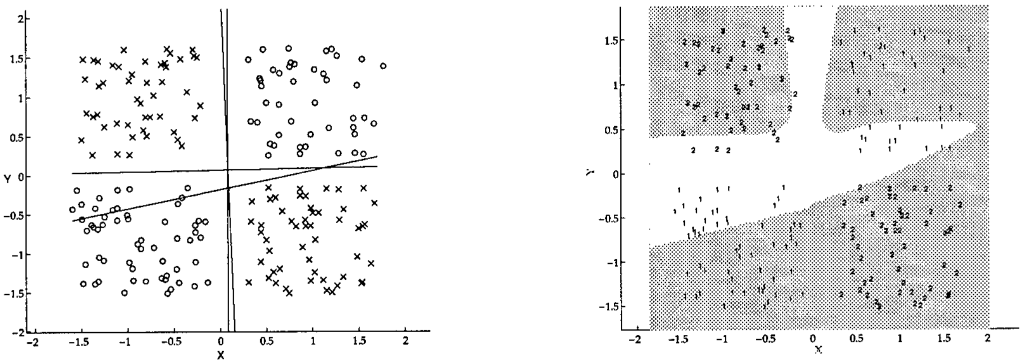
```
while more class pairs $i$ and $j$ do
  $Class_i \leftarrow ClassData$ {Transfer all data pertaining to class $i$}
  $Class_j \leftarrow ClassData$ {Transfer all data pertaining to class $j$}
  $\mathbf{D} \leftarrow distance(Class_i, Class_j)$ {Equation 2.3}
  while $\mathbf{D}_{l,m} \neq \infty \ \forall l, m$ do
    $[row,\ column] \leftarrow min(\mathbf{D})$
    $\mathbf{P}_1 \leftarrow Class_i(row, :)$ {Transfer the data vector}
    $\mathbf{P}_2 \leftarrow Class_j(column, :)$ {Transfer the data vector}
    $\mathbf{W}_k \leftarrow TrainBaumHLNs(\mathbf{P}_1, \mathbf{P}_2)$
    $\mathbf{D} \leftarrow TestForSeparation(\mathbf{D}, W_k)$
  end while
end while
```

be seen, because the plus and minus ϵ hyperplane are very close together. Notice how the nearest neighbors from opposite classes appear to have a hyperplane through them and the presence of a hyperplane normal to the first hyperplane. These instances are circled. In the example, four sets of Baum hyperplanes triplets are added and those were enough to separate every other inter-class data pair in the data set. The TrainHiddenBaum algorithm does not supersede the TrainHiddenNodes algorithm because it is slower and adds more HLNs than are necessary for many data sets. Therefore, it is only used when extension one above fails to completely separate the data..

The SL-MICA algorithm follows the iterative approach described above until the missed data vectors from one iteration are the same as the missed data vectors in the next iteration. The TrainHiddenBaum algorithm is then used. This algorithm is clarified below. Thus, SL-MICA, employing Baum's technique, is guaranteed to correctly classify any training set in general position to 100% accuracy. A proof to this effect is presented in Section 3.2.

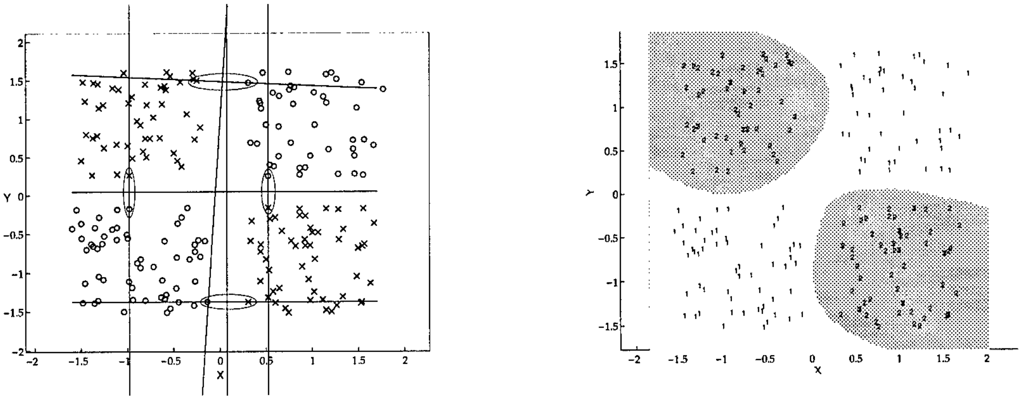
2.6 Relaxation of HLNs

MICA optimizes the HLN placements to maximize training set classification accuracy. This may hinder test set classification accuracy for data sets with overlapping classes. A standard approach in the Backprop world is to start with too many HLNs and prune down. A number of pruning techniques are discussed in [22]. However, many construction



(a) Data trained with TrainHiddenNodes

(b) Classification coloring after training



(c) Data Trained with TrainHiddenBaum

(d) Classification coloring after training

Figure 2.7 TrainHiddenNodes algorithm generates the HLNs in (a) and resulting classification coloring in (b). Then, the same data are trained with TrainHiddenBaum which generates the hyperplanes in (c), which results in a new classification coloring in (d).

Algorithm 8 SL-MICA

```
$\mathbf{NormData} \leftarrow Normalize(\mathbf{Data})$
$\mathbf{SubData} \leftarrow \mathbf{NormData}$
$Accuracy \leftarrow 0$
$SameData \leftarrow False$
while $Accuracy \neq 100 \wedge SameData = False$ do
  $\mathbf{Weights}^1 \leftarrow TrainHiddenNodes(\mathbf{SubData})$ {Call to TrainHiddenNodes Algorithm 2}
  $\mathbf{Weights}^2 \leftarrow TrainOutputNodes\left(\mathbf{Weights}^1, \mathbf{NormData}\right)$ {Call to TrainOutputNodes Algorithm 4}
  $\mathbf{OldSubData} \leftarrow \mathbf{SubData}$
  $[Accuracy, \mathbf{SubData}] \leftarrow TestForAccuracy(\mathbf{NormData}, \mathbf{Weights})$
  if $\mathbf{OldSubData} = \mathbf{SubData}$ then
    $\mathbf{Weights}^1 \leftarrow TrainHiddenBaum(\mathbf{SubData})$
    $\mathbf{Weights}^2 \leftarrow TrainOutputNodes\left(\mathbf{Weights}^1, \mathbf{NormData}\right)$
    $SameData \leftarrow True$
  end if
end while
```

algorithms start with smaller networks and grow the network for best results [4, 39]. SL-MICA does both. First it increases the number of HLNs needed to classify the training set to 100% accuracy, then through relaxation, SL-MICA prunes the least important HLNs to maximize test set accuracy, see Algorithm 9.

The pruning is accomplished by removing one HLN at a time starting with the least important node and moving towards the more important nodes. The importance of the HLNs are measured by the combined size, number of points, of the neighborhoods of the inter-class data pairs used to produce the HLNs. This strategy is predicated by the theory that HLNs that separate the smallest numbers of points are providing the least benefit and that these points are likely in areas of overlapping data. The results show this pruning strategy is effective and keeps the added FLOPS to a minimum.

2.7 Test Results For Single Hidden Layer MLPs (SL-MICA)

Experiments were run to test SL-MICA's different learning schemes and SL-MICA's ability to define an appropriate network architecture for Backprop. There are four data sets in the tests that are separable with one iteration. The spiral data are two streams of

Algorithm 9 Relaxation

```
$NewAccuracy \leftarrow TestAccuracy$
$[NumPointsSorted, Index] \leftarrow Sort(NumPoints)$ {Index is a mapping back to the original order of the data}
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $\mathbf{ReducedWeights}^1 \leftarrow \mathbf{Weights}^1(:, Index(i+1 : NumHLNs))$ {Transfers HLN weight values for all HLN whose importance value ranks higher than $i$}
  $\mathbf{ReducedWeights}^2 \leftarrow FastTrainOutputNodes\left(\mathbf{ReducedWeights}^1, TrainData\right)$ {Call to OLN training algorithm 5}
  $OldAccuracy \leftarrow NewAccuracy$
  $NewAccuracy \leftarrow TestForAccuracy(TestData, \mathbf{ReducedWeights})$
until $NewAccuracy < OldAccuracy$
$i \leftarrow i - 1$
$\mathbf{Weights}^1 \leftarrow \mathbf{Weights}^1(:, Index(i+1 : NumHLNs))$
$\mathbf{Weights}^2 \leftarrow TrainOLN\left(\mathbf{ReducedWeights}^1, Data\right)$
```

data spiraling out from the center [10]. The mesh data form a three-class problem with blocks of data situated adjacent to each other with some class overlap. Iris data form a well known three class problem that attempts to recognize flower types from features measured from the flower's petals [11]. The NIST Optical Character Recognition (OCR) data set is a 10 class problem of the digits zero through nine, using 27 spectral features. There are two data sets that need more than one iteration to separate. The XOR data set is a set of data that mimics a logical XOR problem. The Asynchronous Transfer Mode (ATM) Access Control data set is a two class problem with a large number of samples [15]. The classes are "accept" or "reject" and the features are taken from the statistics of the data packets. The non linear and complex statistical characteristics of the network create a very difficult and intertwined two class problem. The follow seven experiments are run on each data set.

- SL-MICA - SL-MICA without pruning relaxation or reuse
- SL-MICA/relax - SL-MICA with pruning relaxation (to reduce number of HLNs)
- SL-MICA/reuse - SL-MICA without relaxation but with HLN reuse (to prevent unnecessary HLNs)
- Backprop - Backprop using architecture found from SL-MICA (Number of HLNs)

- Backprop/fixed - Backprop using a fixed number of hidden nodes
- Backprop/HLN - Backprop started with the HLN weights found in SL-MICA
- Backprop/HLN/OLN - Backprop started with the HLN and OLN weights found in SL-MICA

Tables 2.1 through 2.7 contain the results of all the experiments for each data set. The results for the Spiral, Mesh, Iris, and XOR data are the average results from 25 test runs. In each test run, a random training and test set were selected prior to the algorithms being run on the data. The OCR test results are the average of ten test runs, while the ATM Access Control is from just one run.

2.7.1 Spiral. Table 2.1 indicates the spiral test results. This data set is used to demonstrate a class of problems where SL-MICA is far superior to that of Backprop. Backprop has difficulty training weights for data sets with areas of low Mean Squared Error (MSE) [10]. The results show that SL-MICA achieves 100% accuracies on the training data, while Backprop's accuracies are approximately 60%, after 300 epochs. Other researcher's results indicate that 150,000 to 200,000 epochs were needed for Backprop to solve the spiral problem, i.e., get a reasonable classification accuracy [10]. SL-MICA does not perform as well on the test set for two reasons. The points in the data set are not very dense and the training and test sets are chosen at random. Therefore, it is not likely that every other point is split between the two data sets. This allows whole arcs to be in one set or the other. Thus, the hyperplanes in SL-MICA may be drawn to cut off whole arcs in the test set. The Backprop/HLN produces an important result. Backprop, given a good set of separating hyperplanes, can successfully learn the output layer weights to separate the data. This is the first result of its kind. The Backprop/HLN/OLN algorithm starts Backprop in an extremely confined area of the weight search space. This algorithm is unlikely to escape its starting valley, but it does act as a local search should there be a smaller minimum to find. It is interesting note that although Backprop lowers the MSE of the weights found in SL-MICA the classification accuracies did not increase. Figure 2.8 shows the results of four of these algorithms on the entire data set.

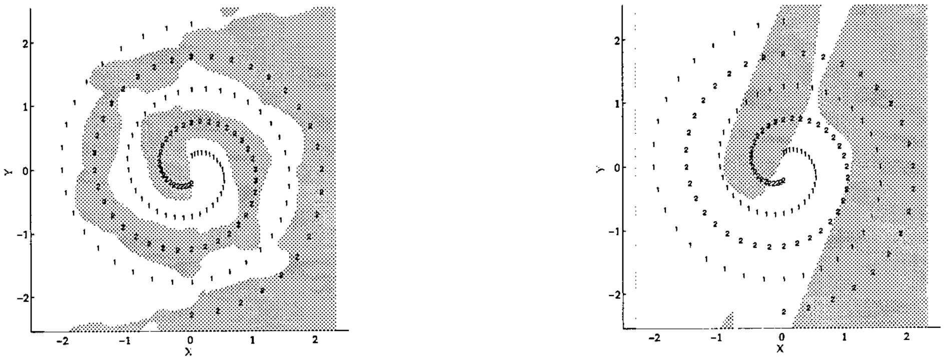
Table 2.1 Average Spiral Data Set Results, 25 Runs

Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	22.88	NA	100%	63.75%	81.87%	2.9136×10^7
SL-MICA/relax	22.24	NA	98.57%	65.42%	81.99%	5.9171×10^7
SL-MICA/reuse	22.88	NA	100%	63.75%	81.87%	2.9136×10^7
Backprop	22.88	300	63.39%	52.17%	57.78%	5.0117×10^7
Backprop/fixed	25	300	64.08%	49.83%	56.96%	5.4548×10^7
Backprop/HLN	22.88	100	99.55%	60.50%	80.03%	3.1547×10^7
Backprop/HLN/OLN	22.88	50	96.61%	62.50%	79.56%	3.6657×10^7

Each algorithm is trained on the same data. SL-MICA required 27 HLNs to correctly classify the data (Figure 2.8(a)). Backprop was also given 27 HLNs (Figure 2.8(b)), but after 300 epochs the MSE mostly oscillated and barely changes, see Figure 2.9 top line. The Backprop/HLN algorithm (Figure 2.8(c)) classifies the data correctly. In this case, Backprop quickly adjusts the output level weights while only slightly modifying the hidden layer weights. As expected, this error is smooth and drops quickly, see Figure 2.9 middle line. Notice that the random weights initially have a lower MSE than the partially seeded weights. The MSE for the fully seeded weights starts at a minimum level and decreases slightly, and the output has only minor changes from SL-MICA's, see Figure 2.8(d).

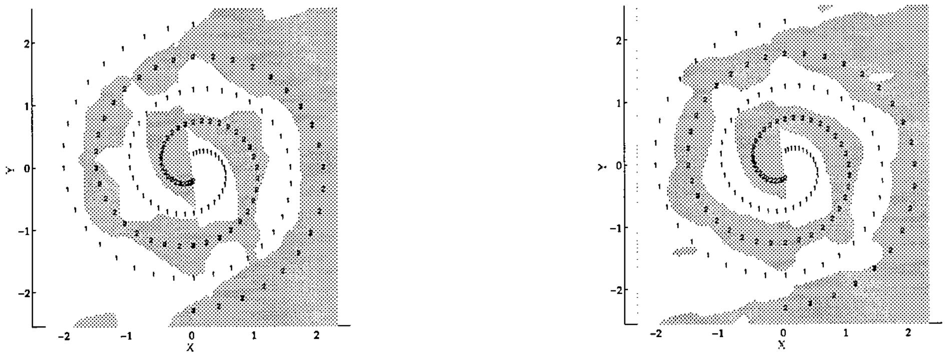
2.7.2 Mesh. Table 2.2 reports the Mesh test results. The data set is used because it favors Backprop. Backprop's strength is defining hyperplanes for data sets that have all high areas of MSE. SL-MICA achieves results slightly worst than Backprop for the test set. SL-MICA with relaxation achieves the same results as backprop on the test set without losing significant accuracy on the training set and achieves an overall accuracy higher than Backprop's. Relaxation, on average, reduced the number of HLNs by two nodes. In one case it reduced the number of HLNs by 7, see Figure 2.10 to see how the output space changes coloring. In contrast to its performance on the spiral data set, the Backprop/HLN/OLN algorithm improves upon SL-MICA's performance.

Figure 2.10 illustrates the decision boundaries before and after relaxation. Before relaxation the boundaries are sharper, see figure 2.10(a) and 2.10(b). After relaxation, see Figure 2.10(c) and 2.10(d), the boundaries more closely resemble Backprop's, see Figure 2.10(e) and 2. 10(f).



(a) MICA on Spiral Data - 100% Accuracy

(b) Backprop on Spiral Data - 61.2% Accuracy



(c) Backprop/HLN on Spiral Data -100% Accuracy

(d) Backprop/OLN on Spiral Data -100% Accuracy

Figure 2.8 The complete x-y coordinate grid is fed into four different MLP networks trained with the complete data set using (a) SL-MICA, (b) Backprop with momentum, (c) Backprop/HLN, which seeds Backprop with the HLN weights from SL-MICA, and (d) Backprop/OLN which seeds Backprop with all the weights from SL-MICA

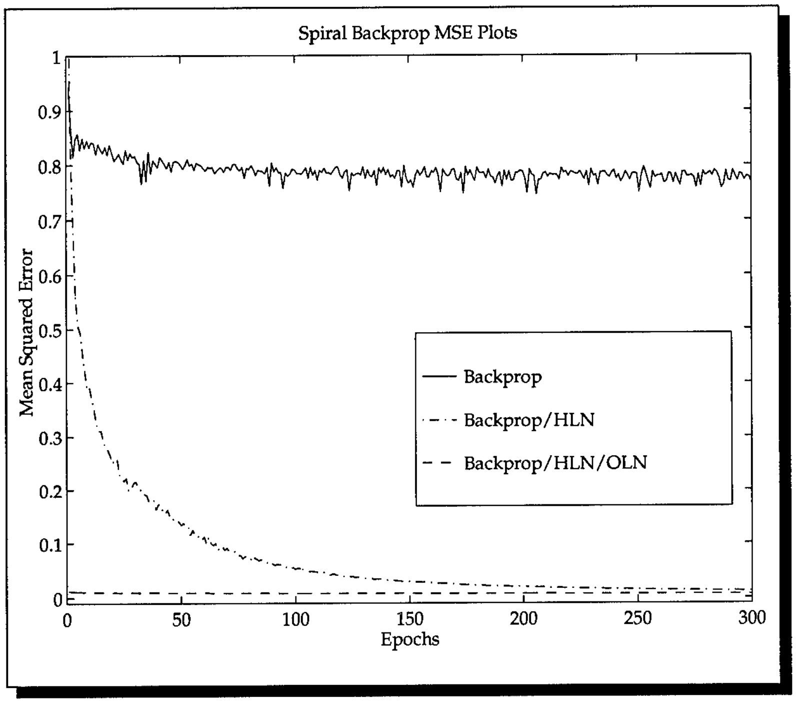


Figure 2.9 Training error for the three Backprop algorithms.

Table 2.2 Average Mesh Data Set Results, 25 Runs

Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	17.24	NA	100%	94.05%	97.02%	1.0436×10^8
SL-MICA/relax	15.28	NA	99.88%	94.59%	97.23%	2.4684×10^8
SL-MICA/reuse	16.04	NA	100%	94.09%	97.04%	1.0525×10^8
Backprop	17.24	300	98.42%	94.69%	96.55%	1.7258×10^8
Backprop/fixed	15	300	98.37%	94.79%	96.58%	1.5101×10^8
Backprop/HLN	17.24	100	99.29%	93.71%	96.5%	1.2104×10^8
Backprop/HLN/OLN	17.24	50	100%	94.21%	97.1%	1.3194×10^8

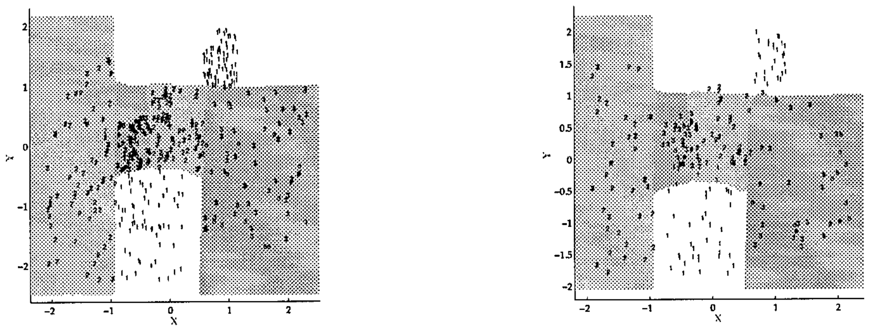
This suggests the the difference in SL-MICA and Backprop is SL-MICA's ability to define the HLNs in areas of low MSE. In addition, this figure shows that the test data is missed because the data were along the boundaries. Relaxation picks up some of the missed points because the new decision boundaries are not as tight to the training data with respect to the class data with the misclassified data.

2.7.3 Iris. Table 2.3 contains the Iris test results. The Iris data highlights the SL-MICA/Relax algorithm's ability to generalize. In the Iris data, classes 1 and 2 are linearly separable as are classes 1 and 3; but classes 2 and 3 overlap. This information is derived for the MICA algorithm. Different choices of the train and test data set can cause the test results to vary between 80% to 100% accuracy. Again, SL-MICA's accuracy is slightly less than Backprop's for the test data, but SL-MICA with Relaxation is two percentage points better than SL-MICA and one percentage point higher than Backprop. The relaxation algorithm removes HLNs used to memorize the overlapping data and improves generalization. In one trial, relaxation reduced the number of HLNs from 10 to 1 and increased test set accuracy from 96% to 100%. The Backprop/HLN/OLN improves upon SL-MICA, but SL-MICA appears to be adept at learning weights near the bottom of local extrema where there is little room for Backprop to maneuver.

Table 2.3 Average Iris Data Set Results, 25 Runs

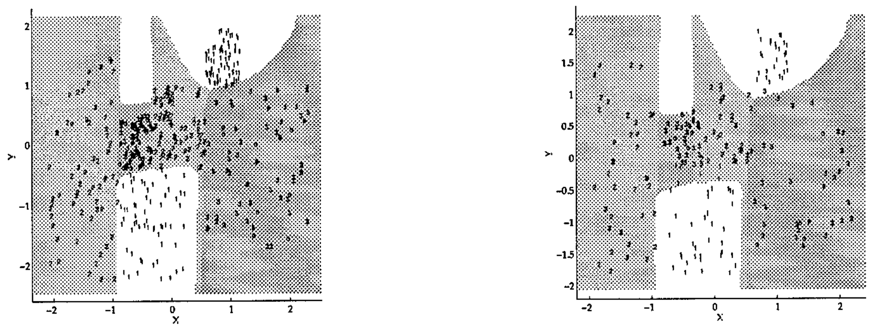
Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	7.08	NA	100%	95.37%	97.69%	1.5462×10^7
SL-MICA/relax	3.52	NA	98.99%	97.28%	98.8%	1.8767×10^7
SL-MICA/reuse	6.08	NA	100%	95.14%	97.57%	1.4979×10^7
Backprop	7.08	300	99.15%	96.16%	97.65%	2.3116×10^7
Backprop/fixed	8	300	99.23%	95.92%	97.58%	2.5895×10^7
Backprop/HLN	7.08	100	100%	95.29%	97.65%	2.1093×10^7
Backprop/HLN/OLN	7.08	50	99.96%	95.45%	97.71%	1.9293×10^7

2.7.4 OCR. Table 2.4 shows the OCR test results. This data set is used for two purposes. With its 10 classes and 3471 vectors, it is a suitable test for stressing the algorithm. It is also a good test of the reuse feature which affords 36 opportunities for reusing existing HLNs, see Equation 2.2. This data test has been a barometer of the evolution of SL-MICA. Originally, the algorithm used a genetic algorithm to find



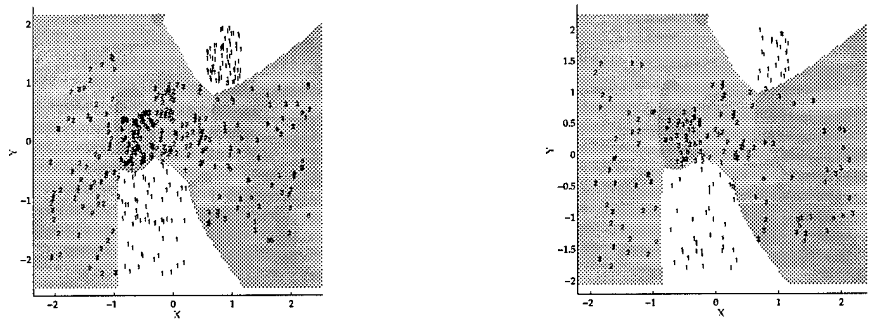
(a) MICA on Train Data - 100% Accuracy

(b) MICA on Test Data - 91.04% Accuracy



(c) MICA/Relax on Train Data - 99.25% Accuracy

(d) MICA/Relax on Test Data - 92.54% Accuracy



(e) Backprop on Train Data - 98.75% Accuracy

(f) Backprop on Test Data - 89.55% Accuracy

Figure 2.10 The complete x-y coordinate grid is fed into three different MLP networks trained using SL-MICA, SL-MICA, and Backprop. These results are overlayed with the training data in (a), (c) and (e) and with the test data in (b), (d), and (f)

the weights for the HLNs. This approach took about 250 HLNs and resulted in test set accuracies in the 90% range. Then using the Ho-Kashyap method and a fixed neighborhood size, the HLNs reduced to around 160 and test accuracies were around 92%. Finally, with the variable size neighborhoods the HLNs reduced further to approximately 60 and the test accuracies rose to the 95% range. The HLN reuse further reduced the number of HLNs to an average of 55.4, with the test set accuracies just about equal with SL-MICA. The relaxation technique is not as effective as reuse in reducing HLNs, but it does achieve a higher test set accuracy. The Backprop algorithm has similarly improved its accuracies as SL-MICA has improved because Backprop is architecture sensitive. As SL-MICA improved its ability to define the upper bound on the number of HLNs needed to shatter the data, Backprop improved its performance by training on a better network architecture. There appears to be a strong correlation at the macro level between the number of HLNs used and the test set accuracies, but only a small correlation at the micro level. That is, nets that have grossly too many or too few HLNs perform much worse than nets with the appropriate number. However, nets that vary by only a small amount of HLNs perform about the same. This is illustrated by the Backprop and Backprop/fixed tests.

Table 2.4 Average OCR Data Set Results, 10 Runs

Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	62.3	NA	100%	95.65%	97.76%	1.4516×10^{10}
SL-MICA/relax	61.2	NA	100%	95.81%	97.91%	1.8569×10^{10}
SL-MICA/reuse	55.4	NA	100%	95.52%	97.76%	1.7283×10^{10}
Backprop	62.3	300	99.63%	96.04%	97.83%	1.8038×10^{10}
Backprop/fixed	60	300	99.70%	95.85%	97.77%	1.7379×10^{10}
Backprop/HLN	62.3	100	99.80%	95.98%	97.89%	1.6359×10^{10}
Backprop/HLN/OLN	62.3	50	100%	96.05%	98.02%	1.7568×10^{10}

2.7.5 XOR. Table 2.5 shows the XOR test results. This data set is used because SL-MICA does not solve it with one pass of the TrainHiddenNodes algorithm. This is the first time SL-MICA needed to iterate on the HLN training. However, the TrainHiddenBaum algorithm was never needed. In a two class problem there are no reuse opportunities, so the SL-MICA/Reuse was the same as SL-MICA. The relaxation technique found a slight improvement for the test set, but since there is no overlap in the data this was to be expected. This XOR data set was well suited for Backprop which consequently did very well, but Backprop did expend an order of magnitude more FLOPS than SL-MICA. The Backprop/HLN/OLN algorithm exhibited a poor performance. Two things may have caused this to occur. Backprop may have been reorganizing the Hyperplanes and either did not have enough epochs to get a good classification result or may have gotten stuck in a local minimum.

Table 2.5 Average XOR Data Set Results, 25 Runs

Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	11.52	NA	100.00%	97.66%	98.83%	3.453×10^6
SL-MICA/relax	11.04	NA	100.00%	97.78%	98.89%	4.663×10^6
SL-MICA/Reuse	11.52	NA	100.00%	97.66%	98.83%	3.453×10^6
Backprop	11.52	300	100.00%	99.82%	99.91%	4.976×10^7
Backprop/fixed	8	300	100.00%	95.80%	99.90%	3.549×10^7
Backprop/HLN	11.52	100	100.00%	99.18%	99.59%	1.700×10^7
Backprop/HLN/OLN	11.52	50	100.00%	72.90%	71.06%	1.143×10^7

2.7.6 ATM Access Control. Table 2.6 reports the ATM Access Control test results. This was a very difficult data set for SL-MICA to classify correctly. After many iterations and employing the TrainHiddenBaum algorithm SL-MICA was able to achieve the 100% classification accuracy goal. However, this resulted in poor test set classification accuracies. Figure 2.11 plots the declining generalization as groups of HLNs are added to increase training set performance. The Backprop/HLN/OLN algorithm has very poor results. This algorithm also had poor results on the XOR data set which also required multiple iterations of training.

Table 2.6 ATM Access Control Data Set Results

Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	599	NA	100.00%	70.30%	85.15%	4.586×10^{11}
SL-MICA/relax	-	-	-	-	-	-
SL-MICA/Reuse	-	-	-	-	-	-
Backprop	599	300	76.63%	74.66%	75.64%	4.483×10^{10}
Backprop/fixed	-	-	-	-	-	-
Backprop/HLN	599	100	85.82%	78.55%	82.19%	1.482×10^{10}
Backprop/HLN/OLN	599	50	25.08%	22.12%	23.60%	4.659×10^{11}

Table 2.7 reports the ATM Access Control test results after one iteration of SL-MICA. Except for the training result on the SL-MICA algorithm all other results are an improvement from the results in Table 2.6. The Backprop/HLN/OLN algorithm performs much better than the multiple iteration run and has the highest training set accuracy among the algorithms run.

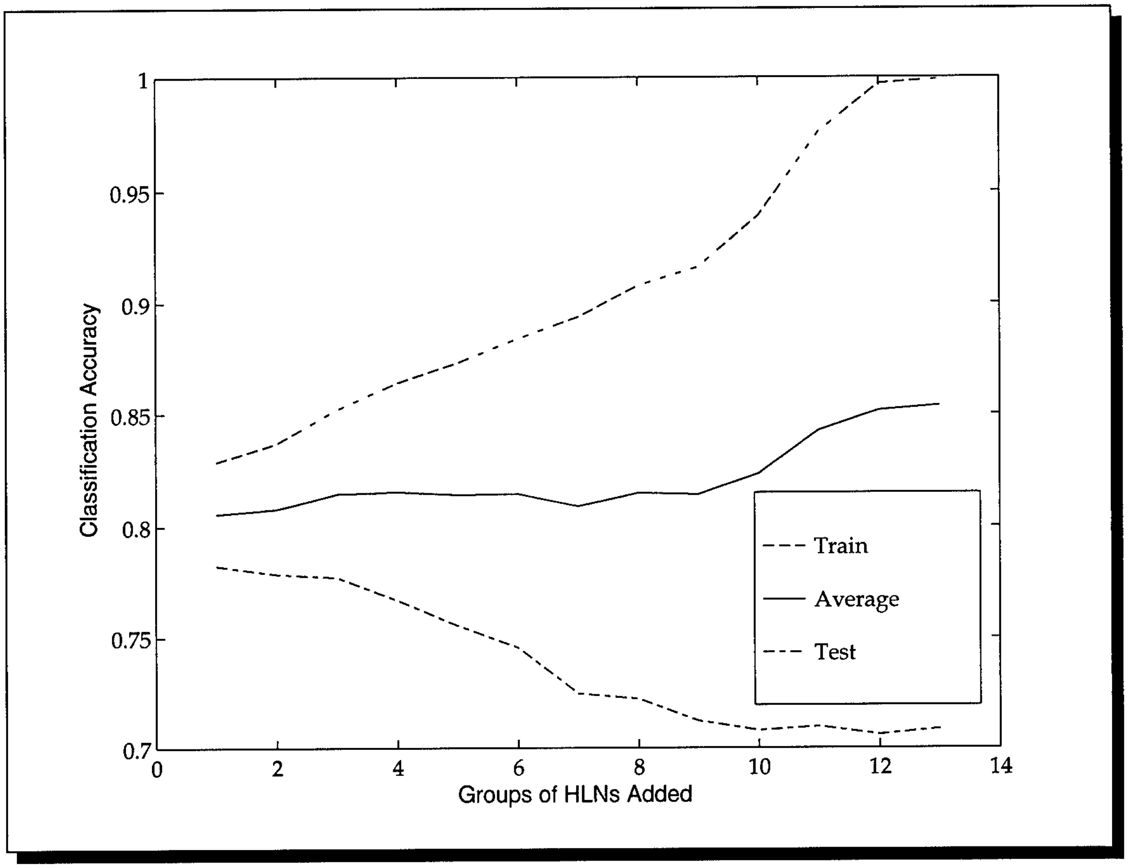


Figure 2.11 The classification accuracies increase with additional HLNs for the training set but, decrease for the test set.

2.7.7 Testing Summary. SL-MICA always achieves 100% recognition accuracies on the training data. When the data set is favorable to Backprop, SL-MICA achieves similar accuracies as Backprop on the test data. When the data set is not favorable to Backprop, SL-MICA achieves far better results than Backprop. The relaxation is effective in increasing the test set accuracies. The reuse algorithm is capable of reducing the number of HLNs needed on larger class data sets. The Backprop/HLN algorithms allows Backprop

Table 2.7 ATM Access Control Data Set Results, 1 Iteration, 2 Runs

Algorithm	Nodes	Epochs	Training	Test	Overall	Flops
SL-MICA	103	NA	82.08%	78.10%	80.09%	1.0056×10^9
SL-MICA/relax	-	-	-	-	-	-
SL-MICA/Reuse	-	-	-	-	-	-
Backprop	103	300	81.67%	78.62%	80.14%	7.7241×10^9
Backprop/fixed	-	-	-	-	-	-
Backprop/HLN	103	100	85.54%	77.07%	81.31%	2.9494×10^9
Backprop/HLN/OLN	103	50	87.40%	76.18%	81.79%	2.2715×10^9

to solve a new class of problems, like the spiral set, with a single hidden layer. The Backprop/HLN/OLN can achieve modest increases over SL-MICA as it performs a local search.

2.8 Conclusions

SL-MICA is both effective and efficient as a learning algorithm for classification problems using MLP networks. SL-MICA realizes the Cybenko network for classification and is particularly well-suited for data sets containing areas of low mean squared error. SL-MICA is also a value-added tool for improving the use of Backprop. It can be used to determine the number of hidden nodes needed, seed the lower level weights, or seed all the weights. In the case of low MSE problems like the spiral data set, SL-MICA is an enabling technology for Backprop to be effective. The next Chapter will proves SL-MICA's ability to classify any data set and the quality of that solution?

III. Theoretical Basis for MICA

3.1 Introduction

Chapter II developed the SL-MICA algorithm for training Single Hidden Layer MLPs. This algorithm first trains as many HLNs as needed to separate all the data in one class from all the data in all other classes. However, this only resulted in a necessary condition for projecting the data into a linearly separable space. Iteratively applying this technique to reduce the missed data vectors, as shown in Section 2.5, still does not guarantee that the data will be projected into a linearly separable space. The sufficient condition is achieved when the missed data are totally isolated with Baum's technique. The reasons for this result will be made clear later in the chapter.

This chapter proves that MICA can produce single hidden layer MLPs that can correctly classify any finite training set, in general position, to 100% accuracy. This proof is by induction, which establishes both the necessary and sufficient conditions for projecting data into a linearly separable space. Then it is shown that MICA meets these conditions. The classification proof is presented in Section 3.2

An important concern of any training algorithm is the quality of solution. That is, how does it compare to the optimal solution. One way to characterize the quality of a classifier's solution is to determine how well it estimates the Bayes optimal discriminant function. For example, Ruck proved that Backprop does a mean squared error approximation of the Bayes optimal discriminant [28]. A similar proof is presented in Section 3.3 to show that MICA is also a mean squared error approximation of the Bayes optimal discriminant. Results of a known Bayes bounded data set is presented in Section 3.4. Lastly, conclusions are presented in Section 3.5.

3.2 Separation Proof

This section contains the proof that MICA can produce an MLP capable of correctly classifying any finite training set. The following definitions are provided to aid in the proof.

Definition 1 (Linearly Separable). *A dichotomy* $\{\mathbf{X}^+, \mathbf{X}^-\}$ *of* $\mathbf{X} \subset \mathbf{R}^d$ *is linearly separable if and only if there exists a weight vector* $\mathbf{w}$ *in* $\mathbf{R}^d$ *and a scalar* t *such that*

$$x \cdot w > t \quad if\ x \in \mathbf{X}^{+} \tag{3.1}$$

$$x \cdot w < t \quad if\ x \in \mathbf{X}^{-}. \tag{3.2}$$

Definition 2 (Homogeneously Linearly Separable). *The dichotomy* $\{\mathbf{X}^{+}, \mathbf{X}^{-}\}$ *of* $\mathbf{X}$ *is said to be homogeneously linearly separable if it is linearly separable with* $t = 0$*.*

Definition 3. *Let* C_n *denote the n dimensional two unit cube* $[-1,1]^n$*.*

Definition 4. *Let* $\mathbf{W}$ *be a p x d+1 matrix of row weight vectors. The function* $\Phi(x, \mathbf{W})$ *is a projection of the vector* $x \in \mathbf{R}^d$ *into a new space* $\mathbf{R}^p$ *defined by the hyperplanes in* $\mathbf{W}$ *and is defined below*

$$\Phi(x, \mathbf{W}) = (\phi(x, w_1), \phi(x, w_2), \cdots, \phi(x, w_p)) \in C_p \tag{3.3}$$

where $\phi(x, w_i) = \tanh([x, 1] \cdot w_i)$.

The following theorems are provided without proof for aid in the proof.

Theorem 1 ([20]). *Let M be a linear subspace of a linear space X, and let* l_m *be a linear functional defined on M. Then there exists a linear functional l defined on X such that l is an extension of* l_m*, that is, l is defined on X and* $l(x) = l_m(x)$ *for all* $x \in M$*.*

Theorem 2 ([20]). *Let M be a proper subspace of a linear space X, and let* x_o *be a point in* $X - M$*. Then there exist a linear functional l on X such that* $l(x) = 0$ *if* $x \in M$ *and* $l(x_o) = 1$*.*

This next theorem is used to illustrate that if data in one space are homogeneously linearly separable, then by adding one new dimension to the space the data are still homogeneously linearly separable with the "same" discriminant function.

Theorem 3. *Let* $\{\mathbf{X}^{+}, \mathbf{X}^{-}\}$ *be an arbitrary dichotomy of* $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n}\} \subset \mathbf{R^d}$ *and let* $\mathbf{W}^1 = \left\{w_1^1, w_2^1, \cdots, w_p^1\right\}$*, where* $w_k^1 \in \mathbf{R}^{d+1}$ *for all* $k = 1, \cdots, p$*. If* $\{\mathbf{X}^{+}, \mathbf{X}^{-}\}$ *is homogeneously linearly separable in the* Φ*-space, i.e. there exists a nonzero* $w^2 \in \mathbf{R}^{p+2}$ *such that*

$$\left[\Phi\left(x_i, \mathbf{W}^1\right), 1\right] \cdot w^2 > 0, \quad for\ all\ x_i \in \mathbf{X}^{+} \tag{3.4}$$

$$\left[\Phi\left(x_j, \mathbf{W}^1\right), 1\right] \cdot w^2 \quad < 0, \quad for\ all\ x_j \in \mathbf{X}^- \tag{3.5}$$

then by adding an arbitrary new hyperplane so that $\hat{\mathbf{W}}^1 = \mathbf{W}^1 \cup \left\{w_{p+1}^1\right\}$, *then there exists* $\hat{w}^2$ *such that*

$$\left[\Phi\left(x_i, \hat{\mathbf{W}}^1\right), 1\right] \cdot \hat{w}^2 \quad > 0, \quad for\ all\ x_i \in \mathbf{X}^+ \tag{3.6}$$

$$\left[\Phi\left(x_j, \hat{\mathbf{W}}^1\right), 1\right] \cdot \hat{w}^2 \quad < 0, \quad for\ all\ x_j \in \mathbf{X}^-. \tag{3.7}$$

Moreover, $\hat{w}^2 = \left[w^2, 0\right]$, *see Figure 3.1.*

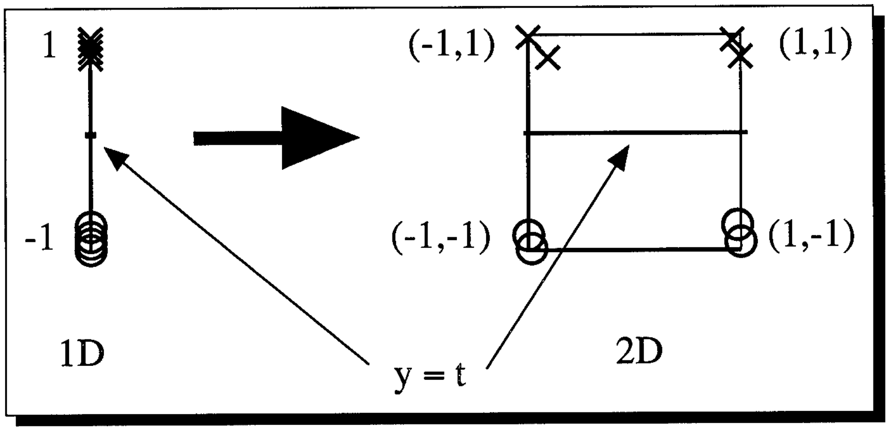


Figure 3.1 Given data that is linearly separable and adding a dimension implies that the data is linearly separable in the new space. Moreover, the equation of the separating hyperplane remains the same.

Proof. Let $\mathbf{X}$ denote the linear space spanned by $\hat{\mathbf{W}}^1$ and let $\mathbf{M}$ denote the linear space spanned by $\mathbf{W}^1$. By definition, M is a linear subspace of X. Since w^2 represents a linear functional on M implies there exist a linear functional $\hat{w}^2$ on $\mathbf{X}$, that is, an extension of w^2. Furthermore, Theorem 1 implies the two linear functionals are equivalent on M, thus $\hat{w}^2 = \left[w^2, 0\right]$. □

The following proof is to show any point in a finite set can be separated from every other point in that set by two hyperplanes.

Theorem 4. *Any point* $x \in \mathbf{X} \subset \mathbf{R}^d$ *can be made linearly separable in* Φ*-space by adding two hyperplanes in* $\mathbf{R}^d$.

Proof. Let $\mathbf{W}^1 = \{w_1^1, \mathrm{w}_2^1\}$, were w_1^1 and w_2^1 represent parallel hyperplanes in the d-dimensional space $\mathbf{R}^d$. Let the distance between the two hyperplanes be arbitrarily small. Furthermore, let the position of these two hyperplanes be such that they "straddle" the point x. There exist an infinite number of pairs of hyperplanes that fit these requirements. Since the set $\mathbf{X}$ is finite there exists at least one orientation of the two hyperplanes that does not "straddle" any another point in $\mathbf{X}$. Therefore, x will be linearly separable from every other point in $\mathbf{X}$ by either w_1^1 or w_2^1. Thus, in C_{d+2}, x will be on a unique vertex and thus be linearly separable. $\square$

The next theorem says that any data in general position can be projected into a linearly separable space with a single hidden layer MLP. This proof will supply the condition for the main result.

Theorem 5. *Any arbitrary dichotomy of a finite set* $\mathbf{X}$ *in general position can be projected into a space where it is homogeneously linearly separable.*

Proof. by induction. It is obviously true for a set of size two.

Assume it is possible for a set of size n. Let $\{\mathbf{X}^+, \mathbf{X}^-\}$ be an arbitrary dichotomy of $\mathbf{X} = \{x_1, x_2, \cdots, x_n\}$, where each $x_i \in \mathbf{R}^d$. Let $\mathbf{W}^1 = \left\{w_1^1, w_2^1, \cdots, w_p^1\right\}$, where each $w_k^1 \in \mathbf{R}^{d+1}$ for all $k = 1, \cdots, p$. Therefore, given $\mathbf{W}^1$, assume there exists $w^2 \in \mathbf{R}^{p+1}$ such that:

$$\left[\Phi\left(x_i, \mathbf{W}^1\right), 1\right] \cdot w^2 \quad > 0, \quad \textit{for all } x_i \in \mathbf{X}^+ \tag{3.8}$$

$$\left[\Phi\left(x_j, \mathbf{W}^1\right), 1\right] \cdot w^2 \quad < 0, \quad \textit{for all } x_j \in \mathbf{X}^-. \tag{3.9}$$

Now show for set of size $n+1$. Let $\mathbf{X}$ increase by one member such that $\hat{\mathbf{X}} = \mathbf{X} \cup \{x_{n+1}\}$. Without Loss Of Generality let $\left\{\hat{\mathbf{X}}^+, \hat{\mathbf{X}}^-\right\}$ be a dichotomy of $\hat{\mathbf{X}}$ such that $\hat{\mathbf{X}}^+ = \mathbf{X}^+ \cup \{x_{n+1}\}$ and $\hat{\mathbf{X}}^- = \mathbf{X}^-$. Need to show there exists a $\hat{\mathbf{W}}^1$ and $\hat{w}^2$ such that:

$$\left[\Phi\left(x_i, \hat{\mathbf{W}}^1\right), 1\right] \cdot \hat{w}^2 \quad > 0, \quad \textit{for all } x_i \in \hat{\mathbf{X}}^+ \tag{3.10}$$

$$\left[\Phi\left(x_j, \hat{\mathbf{W}}^1\right), 1\right] \cdot \hat{w}^2 \quad < 0, \quad \textit{for all } x_j \in \hat{\mathbf{X}}^-. \tag{3.11}$$

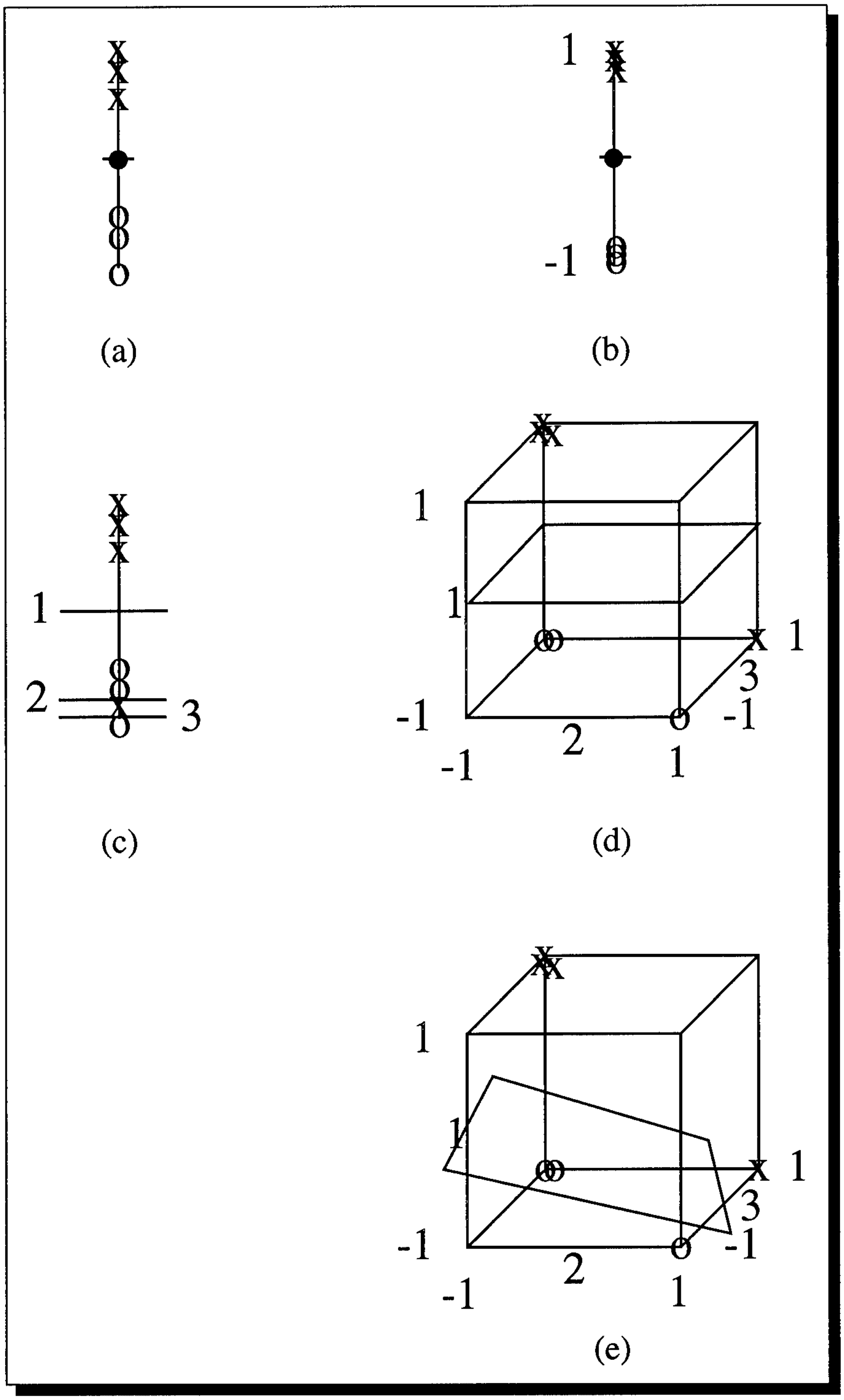
There are three cases to consider. (i) either the $\Phi\left(x_{n+1}, \hat{\mathbf{W}}^1\right)$ is already separable using $\hat{w}^2 = \left(w^2, 0\right)$, (ii) there exists a $\hat{w}^2$ such that $\Phi\left(x_{n+1}, \hat{\mathbf{W}}^1\right)$ becomes separable, or (iii) no $\hat{w}^2$ exists such that $\Phi\left(x_{n+1}, \hat{\mathbf{W}}^1\right)$ becomes separable. Cases (i) and (ii) are trivial. Figure 3.2(a) and (b) illustrates the inductive assumption. For case (iii), since there does not exist a $\hat{w}^2$ where $\Phi\left(x_{n+1}, \hat{\mathbf{W}}^1\right)$ is separable, it implies x_{n+1} is not separable from the members of $\hat{\mathbf{X}}^-$ in the feature space. However, Theorem 4 implies there exists a w^1_{p+1} and w^1_{p+2} that separates x_{n+1} from every point in $\hat{\mathbf{X}}^-$. Figure 3.2(c) shows the addition of hyperplanes 2 and 3 to separate the new point from every one in the other class. Hyperplane 2 is oriented so that points above are on the negative side and points below are on the positive side. Hyperplane 3 is oriented oppositely. Figure 3.2(d) shows the data projected into the new hidden node output space. The new data vector is now on a unique vertex. Let $\hat{\mathbf{W}}^1 = \mathbf{W}^1 \cup \left\{w^1_{p+1}, w^1_{p+2}\right\}$. The addition of these two hyperplanes creates a new hidden node output space, but Theorem 3 implies original dichotomy is still linearly separable and that the equation of the hyperplane that separates the data remains the same as shown in the equations below.

$$\left[\Phi\left(x_i, \hat{\mathbf{W}}^1\right), 1\right] \cdot \left[w^2, 0, 0\right] \quad > 0 \quad for\ all\ x_i \in \mathbf{X}^+ \tag{3.12}$$

$$\left[\Phi\left(x_j, \hat{\mathbf{W}}^1\right), 1\right] \cdot \left[w^2, 0, 0\right] \quad < 0 \quad for\ all\ x_j \in \hat{\mathbf{X}}^- \tag{3.13}$$

Let the area of the space spanned by $\hat{\mathbf{W}}^1$ that is above w^2 be called M_o and the whole space spanned by $\hat{\mathbf{W}}^1$ be call X. Thus, M_o is a proper subspace of X. The point $\Phi\left(x_{n+1}, \hat{\mathbf{W}}^1\right)$ is in $X - M_o$. Theorem 2 says there exists a linear function l such that $l\left(\Phi\left(x_{n+1}, \hat{\mathbf{W}}^1\right)\right) = 1$. This implies there exists a functional that continues to separate the data in the subspace and one new point in the whole space. Figure 3.2(e) depicts a diagram where the separating hyperplane is tilted so that it continues to separate the original data plus one new point on a unique vertex. This implies there exists a $\hat{\mathbf{W}}^1$ and $\hat{w}^2$ such that:

$$\left[\Phi\left(x_i, \hat{\mathbf{W}}^1\right), 1\right] \cdot \hat{w}^2 \quad > 0, \quad for\ all\ x_i \in \hat{\mathbf{X}}^+ \tag{3.14}$$



(a) (b) (c) (d) (e)

Figure 3.2 (a)is an example of linearly separable data. (b) is the linearly separable data projected into the hidden node output space. A new point is added in (c) with two hyperplanes to separate the new data point. (d) shows the new hidden node output space with the two new hyperplanes. (e) shows the new data point on a unique vertex and how the hyperplane can be pivoted to separate the new data point as well as the original data points.

$$\left[\Phi\left(x_j, \hat{\mathbf{W}}^1\right), 1\right] \cdot \hat{w}^2 \quad < 0, \quad for\ all\ x_j \in \hat{\mathbf{X}}^- \tag{3.15}$$

□

Theorem 6. *SL-MICA can correctly classify any data in general position to 100% accuracy.*

Proof. The proof of Theorem 5 provides the basis for this proof. The proof for Theorem 5 says that if two sets of data of opposite class are projected into a linearly separable space and an additional data vector is added to one set that makes the two sets no longer linearly separable, then two new hyperplanes can be added to isolate the new data vector and that is sufficient to project the two data sets into a linearly separable space. The SL-MICA algorithms works by first applying the TrainHiddenNodes algorithm, see Algorithm 2, which places hyperplanes between all data in one class and all the data in the other classes. This is a necessary condition for projecting the data into a linearly separable space, but not sufficient because each point is not isolated. The TrainHiddenNodes algorithm essentially accomplishes the inductive assumption of the proof in Theorem 4. Then each erroneous datum is isolated using Baum's technique. Therefore, via Theorem 5, the data will be projected into a linearly separable space. The Ho-Kashyap algorithm can find a discriminant to separate any data that are linearly separable. Therefore, SL-MICA is guaranteed to correctly classify any training data in general position to 100% accuracy. □

3.3 Bayes Optimal Derivation

Duda and Hart demonstrated how procedures that minimize MSE have the property of approximating the Bayes optimal discriminant function for two class problems [8, pp 154-155]. Ruck used Duda and Hart's two-class problem derivation and applied it to neural networks [26]. Furthermore, Ruck extended Duda and Hart's work and illustrated how Backprop, when training MLP's, approximates the Bayes optimal discriminant in the multi-class case. This section presents a multi-class derivation that proves that MICA produces an MLP that is a minimum MSE approximation of the Bayes optimal discriminant function.

After MICA trains all the HLNs, the data are projected into a new space where the projected data are now linearly separable by class (see proof in Section 3.2). MICA then trains the OLNs by solving a series of two-class problem. In general, MICA solves the weights for output node i by splitting the data into class i and NOT class i and then searching for a discriminant function. By showing that MICA is Bayes optimal in the two class case implies MICA is Bayes optimal in the multi-class case. The following derivation follows Duda and Hart's with only a few deviations.

The Bayes discriminant function is given by

$$g_o(\mathbf{x}) = P(\omega_1 \mid \mathbf{x}) - P(\omega_2 \mid \mathbf{x}) \tag{3.16}$$

where $P(\omega_i \mid \mathbf{x})$ is the probability the true state of nature is ω_i given the observation $\mathbf{x}$. Data samples are distributed according to the probability

$$p(\mathbf{x}) = p(\mathbf{x} \mid \omega_1) P(\omega_1) + p(\mathbf{x} \mid \omega_2) P(\omega_2) \tag{3.17}$$

where $P(\omega_i)$ is the a priori probability of class ω_i and $p(\mathbf{x} \mid \omega_i)$ is the class-conditional probability density function. The mean squared approximation error is given by

$$\epsilon^2(\mathbf{w}) = \int \left[\mathbf{w}^T\mathbf{y} - g_o(\mathbf{x})\right]^2 p(\mathbf{x})\, d\mathbf{x} \tag{3.18}$$

where $\mathbf{y} = \mathbf{y}_i$ and $\mathbf{y}_i = \tanh\left({w_i}^T[\mathbf{x}, 1]\right)$ for all $w_i \in \mathbf{W}$. Therefore, the vector $\mathbf{w}$ that minimizes ϵ^2 also approximates the Bayes optimal discriminant. MICA finds $\mathbf{w}$ by using the Ho-Kashyap algorithm which minimizes $\|\mathbf{Yw} - \mathbf{t}\,\|$, where $\mathbf{Y}$ is the matrix of data vectors,$\mathbf{w}$ is the discriminant vector, and $\mathbf{t}$ is a vector of threshold values. There is one element in $\mathbf{t}$ for each row vector in $\mathbf{w}$. Therefore, a criterion function J_s may be written as

$$J_s(\mathbf{w}) \;=\; \sum_{\mathbf{y}\in\omega_1}\left(\mathbf{w}^T\mathbf{y} - t\right)^2 + \sum_{\mathbf{y}\in\omega_2}\left(\mathbf{w}^T\mathbf{y} + t\right)^2 \tag{3.19}$$

$$= n\left[\frac{n_1}{n}\cdot\frac{1}{n_1}\sum_{\mathbf{y}\in\omega_1}\left(\mathbf{w}^T\mathbf{y}-t\right)^2+\frac{n_2}{n}\cdot\frac{1}{n_2}\sum_{\mathbf{y}\in\omega_2}\left(\mathbf{w}^T\mathbf{y}+t\right)^2\right] \tag{3.20}$$

where n is the number of total points, n_1 is the number of points in class 1, and n_2 is the number of points in class 2. In the Ho-Kashyap method, $\mathbf{t}$ varies individually with each y. Thus, $\mathbf{t}$ is really $t(y)$. Then using the law of large numbers, as n approaches infinity $(1/n)J_s(\mathbf{w})$ approaches

$$\lim_{n\to\infty} J_s(\mathbf{w}) = J(\mathbf{w}) = P(\omega_1)\,E_1\left[\left(\mathbf{w}^T\mathbf{y}-t\right)^2\right]+P(\omega_2)\,E_2\left[\left(\mathbf{w}^T\mathbf{y}-t\right)^2\right] \tag{3.21}$$

with probability one, where

$$E_1\left[\left(\mathbf{w}^T\mathbf{y}-b\right)^2\right] = \int\left(\mathbf{w}^T\mathbf{y}-t\right)^2 p(\mathbf{x}\mid\omega_1)\,d\mathbf{x} \tag{3.22}$$

$$E_2\left[\left(\mathbf{w}^T\mathbf{y}-b\right)^2\right] = \int\left(\mathbf{w}^T\mathbf{y}-t\right)^2 p(\mathbf{x}\mid\omega_2)\,d\mathbf{x}. \tag{3.23}$$

Equation 3.16 can be rewritten as

$$g_0(\mathbf{x}) = \frac{p(\mathbf{x},\omega_1)-p(\mathbf{x},\omega_2)}{p(\mathbf{x})}. \tag{3.24}$$

Now $J(\mathbf{w})$ can be rewritten as follows

$$\begin{aligned}
J(\mathbf{w}) &= \int\left(\mathbf{w}^t\mathbf{y}-t\right)^2 p(\mathbf{x},\omega_1)\,d\mathbf{x}+\int\left(\mathbf{w}^t\mathbf{y}-t\right)^2 p(\mathbf{x},\omega_2)\,d\mathbf{x} && (3.25)\\
&= \int\left(\mathbf{w}^t\mathbf{y}\right)^2 p(\mathbf{x})\,d\mathbf{x}-2t\int\mathbf{w}^t\mathbf{y}g_o(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x}+t^2 && (3.26)\\
&= \int\left[\mathbf{w}^t\mathbf{y}-g_o\right]^2 d\mathbf{x}+\left[t^2-t\int g_o^2(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x}\right] && (3.27)\\
&= \epsilon^2(\mathbf{w})+\left[t^2-t\int g_o^2(\mathbf{x})\,p(\mathbf{x})\,d\mathbf{x}\right]. && (3.28)
\end{aligned}$$

Since the second term in (3.28) is not a function of $\mathbf{w}$, the $\mathbf{w}$ that minimizes J_s also minimizes ϵ^2. The above derivation applies to a two class problem. MICA solves a series of two class problems, one for each OLN. If the solution to each OLN is Bayes optimal, then the aggregate solution of all OLNs is Bayes optimal.

3.4 Bayes Bounding Example

In the preceding section, the relationship with Bayes was established. In this section, an example is presented to demonstrate that relationship. First, a data set with a computable Bayes bound is presented and compared to the results of the SL-MICA algorithm.

3.4.1 Bayes Bound. The ATM Access Control data set is used for this example because it is a two class problem with a lot of data. One technique for Bayes bounding is provided by Fukunaga and Hummels [12]. A good explanation of Fukunaga and Hummels technique can be found in Martin's Thesis [18, pages 26-35], who also developed the code for the following example. Figure 3.3 presents graphically the bounds on the classification accuracy of the ATM Access Control data set. K is the number of clusters. In resubstitution, bottom line, the classifier trains and tests with the same data and hence has an overly optimistic estimation of the error. In leave-One-Out, top line, the classifier uses all but one sample to train and tests with the left out sample. Every point is systematically left out to get a estimate of the error. The Bayes error is bounded by these two lines

3.4.2 Comparison with SL-MICA. The SL-MICA algorithm in Chapter II adds HLNs in groups. Each group contains enough HLNs to separate all the remaining missed classified data. After each group is added, SL-MICA decides whether to add more groups based upon classification error. Figure 3.4 shows the classification accuracy as SL-MICA adds more groups of HLNs. The top line is the classification accuracy of the training data as more groups are added and the bottom line is the classification accuracy of the test data. SL-MICA can force the classification accuracy of the training data to go to 100%, but the classification accuracy of the test data will suffer accordingly. As predicted by the Bayes bound, the classification accuracy of the test data was limited to about 80%, and SL-MICA achieves about 78.5% accurately.

3.5 Conclusions

SL-MICA is a proven algorithm for constructing MLP networks that approximate the Bayes optimal discriminate function. It is guaranteed to achieve 100% accuracies on

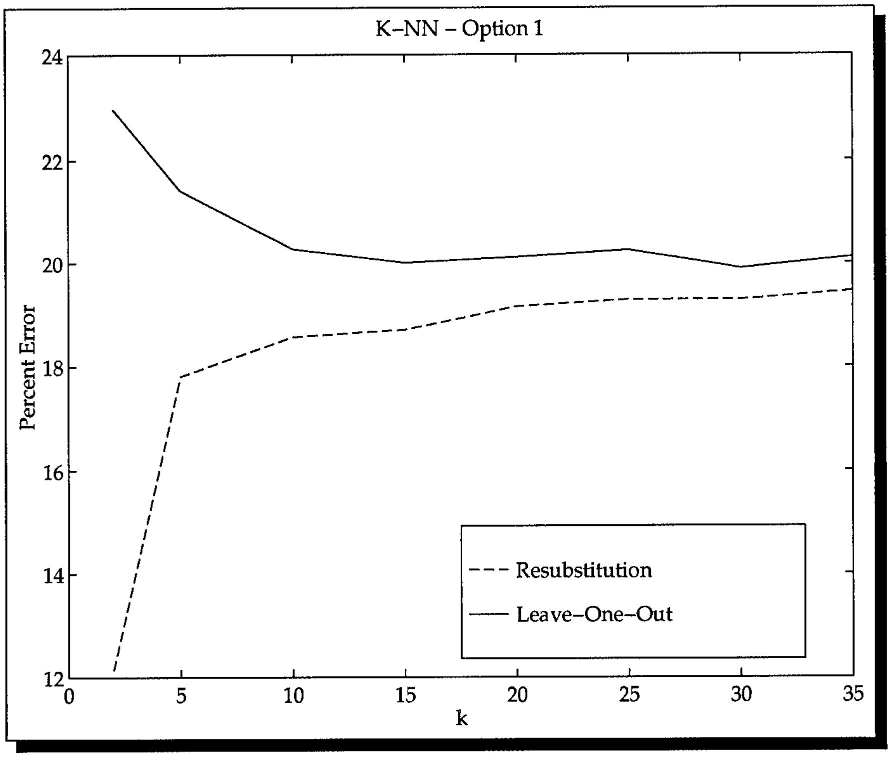


Figure 3.3 The two curves represent the Bayes bounding of the classification error of the ATM Access Control data set.

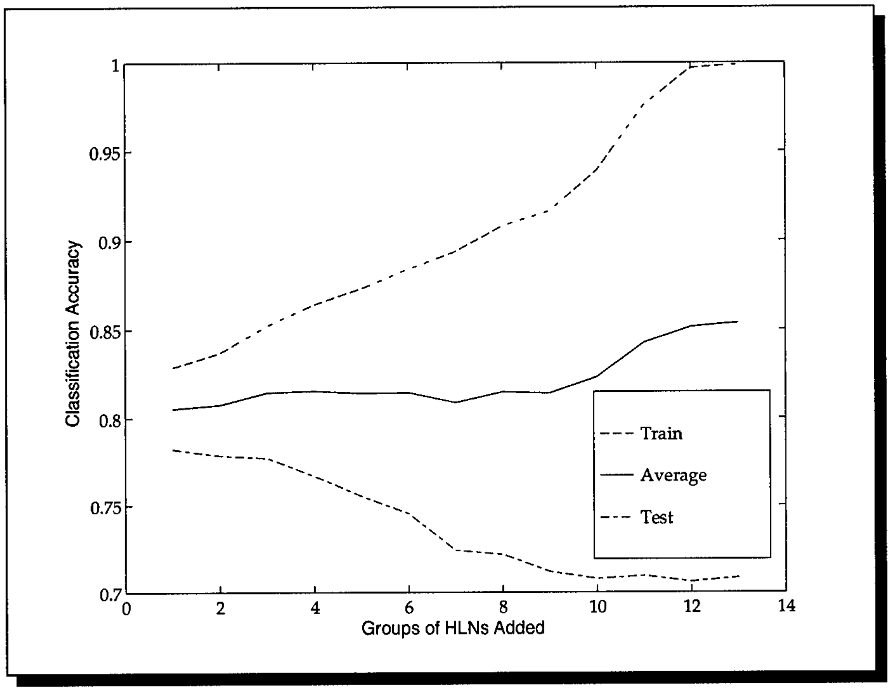


Figure 3.4 The classification accuracies are plotted of the ATM Access Control data set as a function of adding additional HLNs.

the training data and provides excellent generalization results on the test data which are in-line with Bayes Bounding expectations. A widely debated question is: are single or multi layer networks better for generalization? Chapter IV addresses this question with the development of the Multi Layer MICA algorithm.

IV. Construction of Multiple Hidden-Layer MLPs

Chapter II developed a robust algorithm for constructing signal hidden layer MLPs. Then Chapter III explored the effectiveness of the algorithm and the quality of the solutions it produced. The problem with the algorithm, which is demonstrated by the ATM Access Control data set, is it could take a lot of HLNs (almost 500) to get the desired classification accuracy. An alternative approach is to build multi hidden layer MLPs. This Chapter will present a MICA algorithm for constructing a multi hidden layer MLP.

MLP network architecture has been debated for years. There are rules of thumb of deciding how many HLNs can be supported by the data being trained. However, these rules don't apply to MICA since it is not reducing a global error. Also debated is the number of layers needed to get the best results. The Multi Layer MICA (ML-MICA) algorithm builds the network from the ground up. It adds as many HLNs in a layer as needed to attempt to separate the classes, then it only adds additional layers if the data are not linearly separable. The ML-MICA algorithm is presented in Section 4.1. Proofs on classification accuracy and Bayes optimality are in Section 4.2 and Section 4.3 respectively. Section 4.4 presents test results for ML-MICA. This is followed by a discussion of when to use SL-MICA or ML-MICA, which results in a unified MICA algorithm presented in Section 4.5.

4.1 Multi Layer MICA

The block diagram of SL-MICA in Figure 2.1 is still applicable to ML-MICA. The TrainHLNs block, Algorithm 2, is applied to the data followed by the TrainOLNs block, Algorithm 5. If the accuracy of the training data is not 100%, then another hidden layer is added. To add another hidden layer, the data is propagated through the prior hidden layer nodes to produces a new data set. This data set is used as the inputs to the TrainHLNs block and a new layer of HLNs result. The OLNs are retrained and the results are tested. The algorithm quits when enough hidden layers are added to reach 100% classification accuracy of the training data, see Algorithm 10.

Algorithm 10 ML-MICA

```
$\mathbf{NormData} \leftarrow Normalize(\mathbf{Data})$
$Accuracy \leftarrow 0$
$i \leftarrow 0$
while $Accuracy \neq 100$ do
  $i \leftarrow i+1$
  $\mathbf{Weights}^i \leftarrow TrainHiddenNodes(\mathbf{NormData})$ {Call to TrainHiddenNodes Algorithm 2}
  $\mathbf{Weights}^{i+1} \leftarrow TrainOutputNodes\left(\mathbf{Weights}^i, \mathbf{NormData}\right)$ {Call to TrainOutputNodes Algorithm 4}
  $[Accuracy, \mathbf{SubData}] \leftarrow TestForAccuracy(\mathbf{NormData}, \mathbf{Weights})$
  if $Accuracy \neq 100$ then
    $\mathbf{NormData} \leftarrow \left[Tanh\left(\mathbf{Weights}^i * \mathbf{NormData}^t\right), \mathbf{1}\right]$
  end if
end while
```

The ML-MICA algorithm can also be used with relaxation. After each layer is produced, relaxation is applied to eliminate the unimportant nodes. Since the data are going to be re-separated the hyperplanes in each hidden layer, it is possible for the relaxation to be more aggressive. For example, all HLNs that separate only two data pairs may be rejected up front. This approach will prevent over learning in the lower hidden layers.

4.2 Classification Proof for ML-MICA

This section proves that ML-MICA can project any training data in general position, into a linearly separable space by adding enough hidden layers, where the HLNs in each layer are created with the TrainHiddenNodes algorithm.

Theorem 7. *ML-MICA can correctly classify any data in general position to 100% accuracy.*

Proof. The TrainHiddenNodes algorithms puts hyperplanes between every datum in a class and all data in every other class. Every HLN added creates a new dimension in the hidden node output space, whereby it drives the data in one class towards 1 and data in the other class toward -1. This clusters data at the corners of an n dimensional cube. Since the hyperplanes are added to force all class one data towards 1 and class two data towards

-1, the data will gravitate towards opposite corners of the hypercube. Therefore, given enough layers the data will become linearly separable. ☐

4.3 Bayes Optimal Proof for ML-MICA

The proof for SL-MICA in Section 3.3 said that given a set of separating hyperplanes, the training algorithm for the output layer nodes does a mean squared error approximation of the Bayes optimal discriminant function. Thus, the Bayes proof for the multi hidden layer algorithm is exactly the same, since the hidden layer nodes are also predefined when the final layer of output nodes are trained.

4.4 Test Results for ML-MICA

Results are presented for the Iris, Mesh, OCR, XOR, and ATM Access Control data sets. Only the last two sets need additional hidden layers to correctly classify 100% of the training data. Therefore, the algorithm was force to use additional hidden layers when training for the Iris, Mesh, and OCR data sets. The following experiments were run:

- ML-MICA - Multi Level MICA
- ML-MICA/Relax - Multi Level MICA with relaxation

Results are the average of 25 test runs. Each test run randomly creates a training and test set and then runs the algorithms. This experiment forces ML-MICA to create a five hidden layer MLP, but the intermediate results are saved so the classification accuracies can be tracked with the addition of each layer.

4.4.1 Iris. Table 4.1 presents the results for the experiments on the Iris data. The number of HLNs needed to separate the data decreases with each hidden layer. This is because the data are being better organized with each layer and thus the class data are being projected to the same area of the hidden layer node output space. The minimum number of nodes possible without using relaxation is related to the number of classes and is given by Equation 2.2. For a three class problem that number is three because of the pair-wise separations. All the accuracy values are very close, and are not statistically different. Without relaxation a three hidden layer net gives the best generalization then decreases slightly with each additional layer. With relaxation the best generalization is with one hidden layer. It is good that adding unneeded layers does not drastically effect generalization, but it is evident that they do change output space slightly.

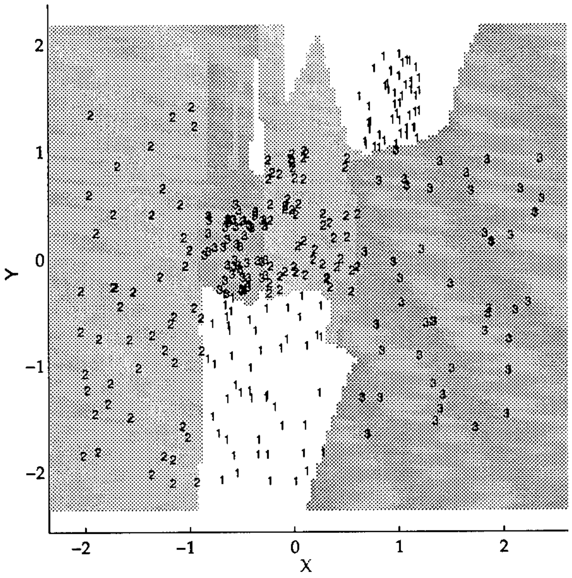
Table 4.1 Iris Data Set Results, 25 Runs

Algorithm	Layer	Training	Test	Overall	# HLNs
ML-MICA	1	100.00%	95.41%	97.71%	4.72
	2	100.00%	95.31%	97.65%	4.20
	3	100.00%	95.47%	97.73%	3.76
	4	100.00%	95.36%	97.68%	3.48
	5	100.00%	95.31%	97.65%	3.48
ML-MICA/Relax	1	99.31%	96.64%	97.97%	2.92
	2	99.15%	96.53%	97.84%	2.64
	3	99.15%	96.43%	97.79%	2.00
	4	99.25%	96.32%	97.79%	1.80
	5	99.52%	96.32%	97.92%	2.16

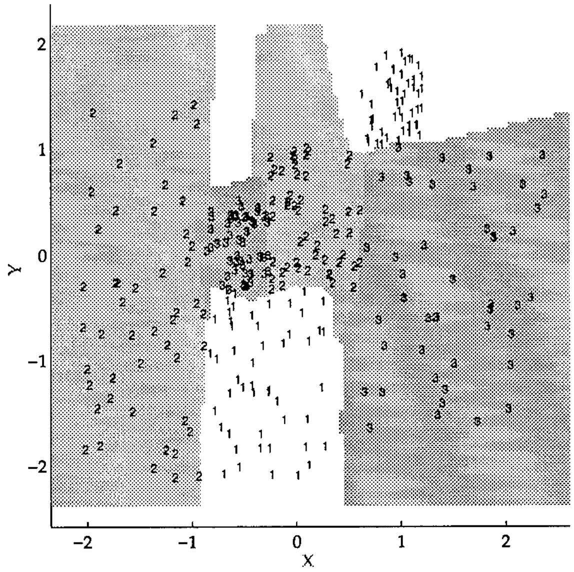
4.4.2 Mesh. Table 4.2 presents the results for the experiments on the Mesh data. Again the number of HLNs to separate the data decreases with each hidden layer added. This data set reaches the lower bound on the number of HLNs needed to separate the data at the third layer. This implies that all the data in each class are projected to a similar area in the hidden node output space. The training classification accuracy of less than 100% is because the number of iterations for training the OLNs was set to one because the multiple layers will eventually get to 100%. Without relaxation the five layer net had the best results. With relaxation a one hidden layer net has the best generalization, but a two layer net has the best overall accuracies. Then the accuracies drop off slightly and remain statistically the same.

Figure 4.1 colors the output space for a one and two hidden layer network to show the difference. Even though the one hidden layer net separates the data, a two hidden layer net seems better able to produce decision boundaries that adhere to the structure of the data.

4.4.3 OCR. Table 4.3 presents the results for the experiments on the OCR data. A strange result occurred in that the number of HLNs per layer increased. A review the



(a) Network output for a one hidden layer network



(b) Network output for a two hidden layer network

Figure 4.1 The color maps indicate the class for each pixel in the output space (a) is for a one hidden layer network and (b), is for a two hidden layer network.

Table 4.2 Mesh Data Set Results, 25 Runs

Algorithm	Layer	Training	Test	overall	# HLNs
ML-MICA	1	99.97%	93.68%	96.83	14.96
	2	100.00%	93.99%	96.99%	3.20
	3	100.00%	93.97%	96.99%	3.00
	4	100.00%	94.01%	97.01%	3.00
	5	100.00%	94.03%	97.01%	3.00
ML-MICA/Relax	1	99.88%	94.24%	97.06%	13.8
	2	100.00%	94.21%	97.11%	3.56
	3	100.00%	94.03%	97.01%	3.04
	4	100.00%	94.07%	97.03%	3.00
	5	100.00%	94.12%	97.06%	3.00

data showed that of the ten test runs, an anomalous event (increased HLNs per layer) occurred in two test runs. However, the increase was so large that it effected that average in a dramatic way. It is not clear why this happened. Without relaxation the one hidden layer net had the best results. Similarly, with relaxation the one hidden layer net had the best results.

Table 4.3 OCR Data Set Results, 10 Runs

Algorithm	Layer	Training	Test	overall	# HLNs
ML-MICA	1	100.00%	95.14%	97.57%	51.9
	2	100.00%	94.83%	97.41%	51.3
	3	100.00%	94.65%	97.33%	51.3
	4	100.00%	94.54%	97.27%	58.5
	5	100.00%	94.52%	97.26%	66.6
ML-MICA/Relax	1	100.00%	95.22%	97.61%	51.1
	2	100.00%	95.04%	97.52%	50.4
	3	100.00%	94.79%	97.39%	51.1
	4	100.00%	94.64%	97.32%	58.8
	5	100.00%	94.48%	97.24%	66.0

4.4.4 XOR. Table 4.4 presents the results for the experiments on the XOR data. Here the number of HLNs to separate the data decreases with each hidden layer. Notice after two hidden layers the data are linearly separable and one HLN is needed to separate the data. Without relaxation the three hidden layer net had the best results. With relaxation a two hidden layer net has the best generalization. This is the first instance where relaxation did not improve the generalization.

Table 4.4 XOR Data Set Results

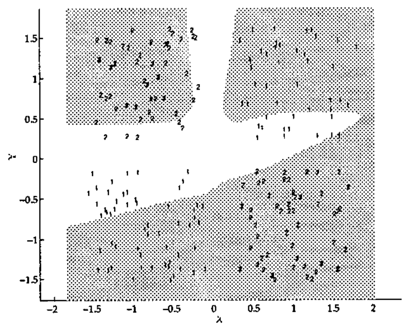
Algorithm	Layer	Training	Test	overall	# HLNs
ML-MICA	1	68.84%	65.32%	67.08%	4.12
	2	100.00%	99.86%	99.93%	2.92
	3	100.00%	99.88%	99.94%	1.00
	4	100.00%	99.88%	99.94%	1.00
	5	100.00%	99.88%	99.94%	1.00
ML-MICA/Relax	1	69.60%	96.28%	67.94%	3.76
	2	100.00%	99.72%	99.86%	2.64
	3	100.00%	99.72%	99.86%	1.00
	4	100.00%	99.72%	99.86%	1.00
	5	100.00%	99.72%	99.86%	1.00

Figure 4.2 colors the output space for a one, two, and three hidden layer network to show the difference. A one hidden layer net does not separate the data, a two hidden layer net produces good decision boundaries that adhere to the structure of the data, and a three hidden layer network is almost exactly the same as a two hidden layer net. This last result is expected since the data, after two hidden layers, could be separated with one HLN.

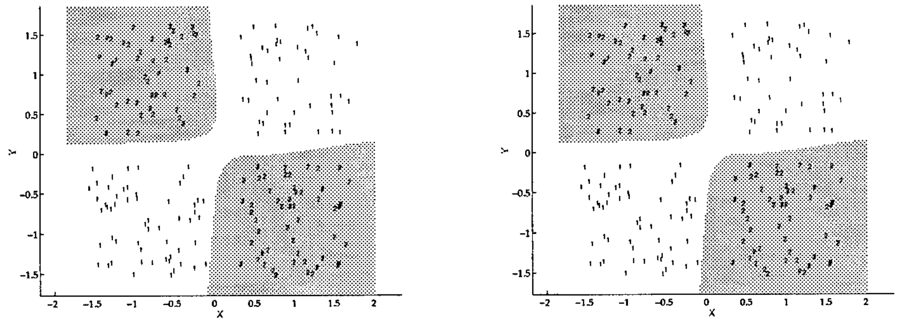
4.4.5 ATM Access Control. Table 4.5 presents the results for the experiments on the ATM Access Control data. These results are from one test run, where the data was randomly split into a training and test set. This experiment lets ML-MICA decide the number of layers needed to obtain 100% classification accuracy on the training data, but the intermediate results are saved so the classification accuracies can be tracked with the addition of each layer. It takes ten layers for ML-MICA to classify the training set at 100%, but the test set accuracies decline with each layer. See Figure 4.3 for a plot of the accuracies as a function of layers. The decline is not as pronounced using relaxation, but the best network for this data set is a single hidden layer MLP.

4.5 SL-MICA verses ML-MICA

One goal of training an MLP is for it to generalize well. Good generalization is achieved when the test set accuracies rival that of the training set. Figures 2.11 and 4.3 show the decline in test set accuracies with each iteration of the respective algorithms for



(a) Network output for a one hidden layer network



(b) Network output for a two hidden layer network

(c) Network output for a three hidden layer network

Figure 4.2 The color maps indicate the class for each pixel in the output space, where (a) is for a one hidden layer network, (b), is for a two hidden layer network and (c), is for a three hidden layer network.

Table 4.5 ATM Data Set Results, 1 Run

Algorithm	Layer	Training	Test	overall	# HLNs
ML-MICA	1	81.78%	79.50%	80.64%	104
	2	87.78%	68.23%	78.01%	42
	3	90.78%	66.26%	78.52%	32
	4	92.77%	64.84%	78.81%	32
	5	95.58%	69.72%	82.65%	30
	6	96.95%	68.91%	82.93%	29
	7	98.71%	69.07%	83.89%	30
	8	99.68%	69.08%	84.38%	12
	9	99.94%	69.16%	84.55%	4
	10	100.00%	69.05%	84.53%	3
ML-MICA/Relax	1	81.78%	79.59%	80.69%	103
	2	85.96%	76.77%	81.36%	42
	3	91.84%	74.36%	83.10%	35
	4	94.27%	72.35%	83.31%	30
	5	96.03%	72.24%	84.13%	29
	6	97.43%	72.22%	84.82%	25
	7	98.12%	72.67%	85.39%	23
	8	99.23%	72.40%	85.81%	23
	9	99.60%	72.69%	86.15%	12
	10	99.95%	72.32%	86.14%	7
	11	100.00%	72.43%	86.21%	2

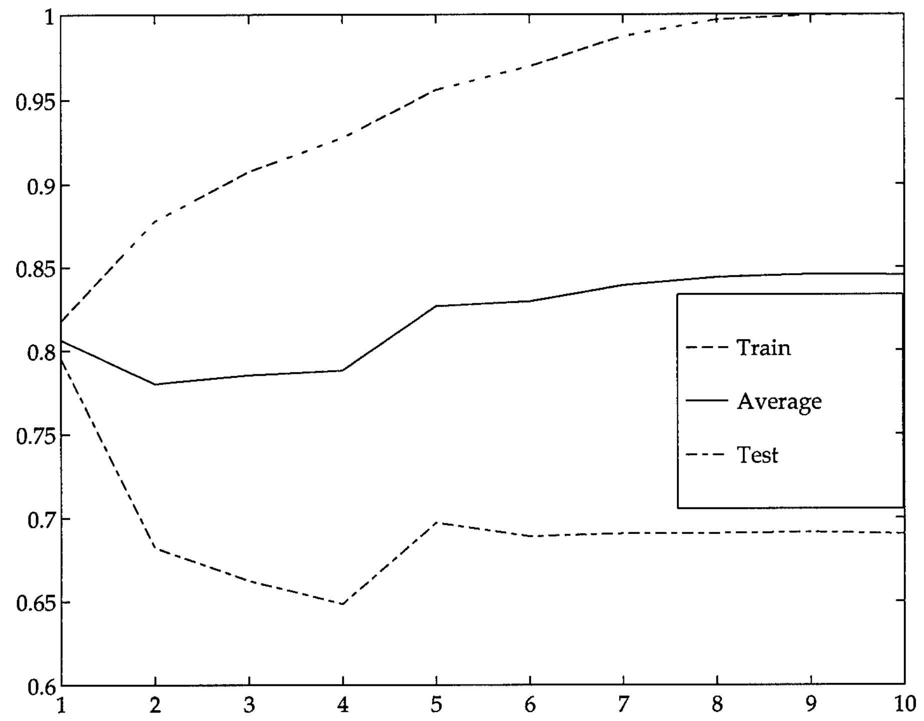


Figure 4.3 The classification accuracies increase for the training set but decrease for the test set.

the ATM data. Therefore, it would be prudent to stop adding nodes or layers if the test set accuracies decrease. To know whether to add more HLNs to the existing hidden layer or to add a new layer is determined by trying both and testing for the best generalization. Also, even if the training set is 100% accurate, more layers may be beneficial for the test set. Therefore, additional layers are added until the test set accuracy stops improving. This completes the MICA algorithm by combining SL-MICA and ML-MICA in a unified algorithm called MICA, see Algorithm 11

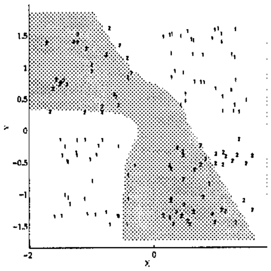
Figure 4.4 plots the classification colorings for intermediate MLPs created by MICA. Figure 4.4(a) is the classification coloring after a single pass of TrainHiddenNodes and TrainOutputNodes. Then MICA needs to decide if adding more nodes to a layer is better than adding another layer. Figures 4.4(b) and 4.4(c) shows the results of these two experiments. MICA selects the architecture used to generated Figure 4.4(b) because is had slightly higher classification accuracies. Then the same tests are performed again in Figures 4.4(d) and 4.4(e). The architecture used for Figures 4.4(d) wins out and additional nodes are added to the same layer instead of growing an additional layer. Since the training set is now classified at 100% accuracy, there is not a choice between adding more nodes to a layer or adding an additional layer. Therefore, only an additional layer is added, but the accuracies of the test set did not improve, so MICA quits.

4.6 Test Results for MICA

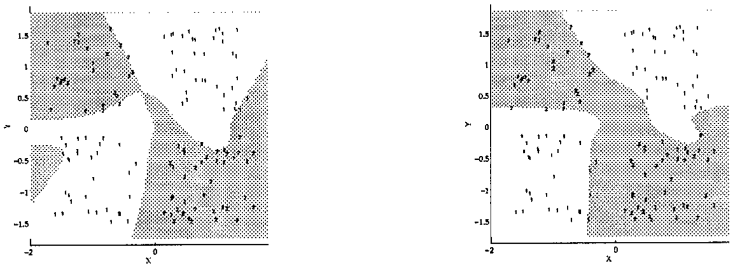
MICA is run on the XOR, Mesh, OCR, and ATM data sets. Since a test set is used to help the algorithm determine when to quit, part of the data set is sequestered in a validation set. Each data set is divided into thirds for training, testing, and validating the network. Table 4.6 presents the initial and final training accuracies for the three data sets. The XOR and Mesh test results are the average of 25 test runs. The OCR result is the average of five test runs and the ATM is from 1 test run. The XOR data set demonstrated great improvement from initial to final results. In 24 of the runs, MICA found that a two hidden layer net was the best solution. The other case resulted in MICA finding a three hidden layer net that generated the best results. For the Mesh data, MICA did not find a consensus in the network layers. Most test runs resulted in a one or a two hidden layer

Algorithm 11 MICA

```
$\mathbf{TrainData} \leftarrow Normalize\,(\mathbf{TrainData})$
$\mathbf{TestData} \leftarrow Normalize\,(\mathbf{TestData})$
$\mathbf{Weights}^1 \leftarrow TrainHiddenNodes\,(\mathbf{TrainData})$ {Algorithm 2}
$\mathbf{Weights}^2 \leftarrow TrainOutputNodes\left(\mathbf{Weights}^i, \mathbf{TrainData}\right)$ {Algorithm 4}
$[TrainAccuracy, \mathbf{SubtrainData}] \leftarrow TestForAccuracy\,(\mathbf{TrainData}, \mathbf{Weights})$
$TestAccuracy \leftarrow TestForAccuracy\,(\mathbf{TestData}, \mathbf{Weights})$
$OldTestAccuracy \leftarrow 0$
$layer \leftarrow 1$
while $TestAccuracy \neq 100\% \wedge TestAccuracy > OldTestAccuracy$ do
  if $TestAccuracy \neq 100$ then
    $\mathbf{WideWeights}^1 \leftarrow TrainHiddenNodes\,(\mathbf{SubTrainData})$
    $\mathbf{WideWeights}^1 \leftarrow \mathbf{Weights}^{layer} \cup \mathbf{WideWeights}^1$
    $\mathbf{WideWeights}^2 \leftarrow TrainOutputNodes\left(\mathbf{WideWeights}^1, \mathbf{TrainData}\right)$
    $[WideTrainAccuracy, \mathbf{SubtrainData}] \leftarrow TestForAccuracy\,(\mathbf{TrainData}, \mathbf{WideWeights})$
    $WideTestAccuracy \leftarrow TestForAccuracy\,(\mathbf{TestData}, \mathbf{WideWeights})$
  end if
  $\mathbf{TempTrainData} \leftarrow \left[Tanh\left(\mathbf{Weights}^{layer} * \mathbf{TrainData}^t\right), \mathbf{1}\right]$
  $\mathbf{TempTestData} \leftarrow \left[Tanh\left(\mathbf{Weights}^{layer} * \mathbf{TestData}^t\right), \mathbf{1}\right]$
  $\mathbf{HighWeights}^1 \leftarrow TrainHiddenNodes\,(\mathbf{TempTrainData})$
  $\mathbf{HighWeights}^2 \leftarrow TrainOutputNodes\left(\mathbf{Weights}^{layer}, \mathbf{TempTrainData}\right)$
  $[HighTrainAccuracy, \mathbf{SubtrainData}] \leftarrow TestForAccuracy\,(\mathbf{TempTrainData}, \mathbf{HighWeights})$
  $HighTestAccuracy \leftarrow TestForAccuracy\,(\mathbf{TempTestData}, \mathbf{HighWeights})$
  $OldTestAccuracy \leftarrow TestAccuracy$
  if $(HighTrainAccuracy > OldTestAccuracy) \vee (WideTrainAccuracy > OldTestAccuracy)$
  then
    if $HighTrainAccuracy > WideTestAccuracy$ then
      $layer \leftarrow layer + 1$
      $\mathbf{Weights}^{layer} \leftarrow \mathbf{HighWeights}^1$
      $\mathbf{Weights}^{layer+1} \leftarrow \mathbf{HighWeights}^2$
      $\mathbf{TrainData} \leftarrow \mathbf{TempTrainData}$
      $\mathbf{TestData} \leftarrow \mathbf{TempTestData}$
      $TrainAccuracy \leftarrow HighTrainAccuracy$
      $TestAccuracy \leftarrow HighTestAccuracy$
    else
      $\mathbf{Weights}^{layer} \leftarrow \mathbf{Weights}^{layer} \cup \mathbf{WideWeights}^1$
      $\mathbf{Weights}^{layer+1} \leftarrow \mathbf{WideWeights}^2$
      $TrainAccuracy \leftarrow WideTrainAccuracy$
      $TestAccuracy \leftarrow WideTestAccuracy$
    end if
  else
    $TestAccuracy \leftarrow OldTestAccuracy$
  end if
end while
```

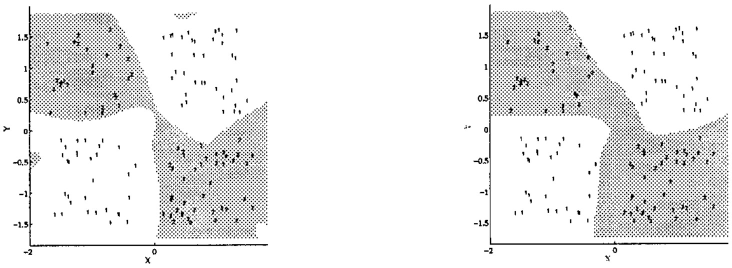


(a) MLP after initial Network Creation



(b) MLP after adding more HLNs to current layer

(c) MLP after add a second hidden layer



(d) MLP after adding more HLNs to current layer

(e) MLP after add a second hidden layer

Figure 4.4 The initial training of MICA results in the classification coloring in (a). Then MICA experiments with adding nodes to the same layer in (b), and adding a new layer of nodes in (c). MICA chooses to add the nodes to the same layer and then repeats the experiments in (d) and (e). Again, adding to the same layer is deemed the best

net, with one case being a four hidden layer net. The change is classification accuracy is small, but the larger net is justified since the validation accuracies also improved. Whereas in the OCR data, the test set improved from initial to final, but the validation set did not. In this case, it may be better to stick with the smaller size network.

Table 4.6 MICA Test Results

Data Set	Layer	# HLNs	Training	Test	Validate
Initial	1		58.23%	56.47%	55.91%
XOR	2.04	6.84	100.00%	99.68%	99.70%
Initial	1		100.00%	90.81%	91.96%
Mesh	1.48	14.44	100.00%	91.56%	92.12%
Initial	1		100.00%	94.16%	94.08%
OCR	2	170	100.00%	94.58%	93.98%
Initial	1		83.00%	76.58%	78.17%
ATM	1	91.4	83.38%	76.71%	78.04%

4.7 Conclusions

ML-MICA proved it could generate Bayes optimal MLPs with 100% training set classification accuracy. ML-MICA generated good test results and demonstrated how multiple layers can improve generalization. Combining the SL-MICA algorithm from Chapter II and ML-MICA from this chapter produced the MICA algorithm which dynamically allocates nodes in a layer or add nodes in a new layer, depending upon the training data set. The MICA algorithm inherits the characteristics of Bayes optimality from SL-MICA. Chapter V will change direction and focus on Feature selection.

V. Feature Selection

5.1 Introduction

There are two methods for reducing the number of weights in a network. First, is to reduce the number of features. Some features may better separate the data than other features, in fact, some features may even hinder separation. Reducing features can not only improve classification, but reduce the net size and speed up training [14]. Second, HLNs can be pruned. MICA adds enough HLNs to construct the network to achieve a given classification accuracy. However, it may be possible to do equally as well with less nodes, especially since MICA separated by class and may add redundant HLNs. Pruning HLNs may also increase generalization by removing HLNs used to memorize data in overlapping areas.

The construction techniques used in MICA yield a novel approach for feature selection. Since MICA trains HLNs to separate class pairs, it can also select features that best separate those class pairs. This unique approach afforded by MICA can improve the performance of many saliency metrics. The feature saliency metric used here is a variation of the decision boundary based saliency metric [16,17]. Results show that the saliency of the features can change drastically depending on which classes are being separated. Therefore, for each pair of classes being separated, only the most salient features are used and the weights connecting the rest of the features are set to zero. The theory of this approach and algorithms to implement are presented in Section 5.2.

Section 5.3 presents a similar technique to that of feature selection for use in HLN pruning. The outputs of the HLNs are considered the features and their saliency is rated in how well they help the OLNs separate the data. Only the most salient HLNs are used to train the OLNs with the other weights set to zero. The experimental results for feature selection and HLN pruning are in Section 5.4 and concluding remarks are in Section 5.5.

5.2 Feature Saliency

5.2.1 Saliency. Feature selection is a well-studied topic [21,26,27,34]. Consider two approaches to feature selection. One approach is to select features by searching for the best combination of the features in terms of how they effect classification accuracy. A second approach is to first rate the features and then select based on that rating. The process of rating features is known as feature saliency. Ruck's saliency uses the partial derivative of the output with respect to each feature [26]. The greatest change implies the most salient features. Tarr sums the magnitude of the weights from each feature to each HLN [34]. The greater the magnitude the more salient the feature.

The MICA Feature Selection (MICA-FS) technique uses a saliency metric based on decision boundaries [16,17] and a metric by Wilson [38]. Decision boundary analysis implies a priori classification of the data. Then one can find the decision boundary between any two data vectors of opposite class. The direction of maximum separation at that point along the decision boundary is the vector normal to decision boundary. Figure 5.1 illustrates this point.

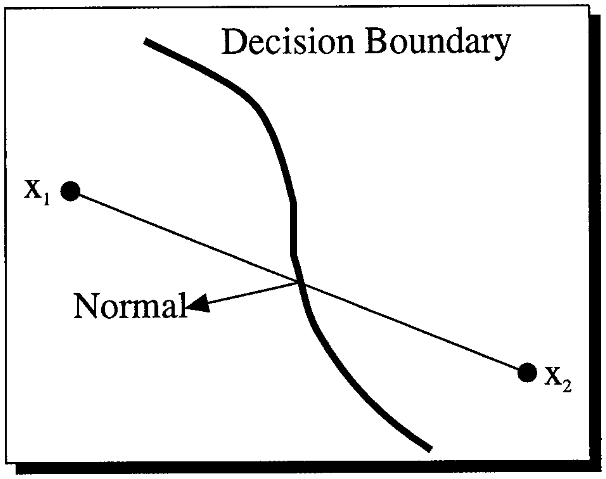


Figure 5.1 The direction of maximum separation is in the direction of the Normal to the decision boundary where the line connecting the two points cross the decision boundary.

The saliency metric sums the outer products of the normals, which produces a matrix. The outer product matrix can be consider an operator over vectors [9]. Consider Equation 5.1

$$\left[v \cdot v^t\right] \cdot x \tag{5.1}$$

where v is a normal vector and x is a vector. The communitive law allows the order of the products to change as follows:

$$\boldsymbol{v} \cdot \left[\boldsymbol{v}^t \cdot x\right]. \tag{5.2}$$

The inner product of two normalized vectors is the cosine of the angle between them, the inner product $\boldsymbol{v}^t \cdot x$ results in a scalar. If $\boldsymbol{v}$ and x are orthogonal the product will be zero and if they are parallel the product will be one. Thus, the outer product of a normalized vector acts as an operator to project x onto $\boldsymbol{v}$. An Effective Decision Boundary Feature Matrix (EDBFM) is created by averaging all the decision boundary normals together. The resulting matrix will hold information about the direction of maximum variance [9]. Calculating the eigenvectors of the EDBFM yields the orthonormal basis set for the directions of maximum separation. The eigenvalues of the EDBFM yield the magnitude of the eigenvectors. Thus, if the maximum direction of separation coincides with an axes of a feature, one can surmise that that feature is most important.

MICA produces unique information that allows MICA-FS to short cut the process of creating the EDBFM. Instead of finding and calculating the normal vectors all along the decision boundary, the vectors normal to the hyperplanes are used, which are the weights of hyperplanes without the bias weight. Then each normal vector is weighted by the number of inter-class data pairs it separates. Recall that MICA trains HLNs to separate inter-class data pairs and their respective neighborhoods. The number of data vectors in the neighborhoods are summed together and used as the weighting for each normal vector. Thus, the EDBFM is calculated from:

$$EDBFM = \frac{\sum_{i=1}^{n} \frac{\mathbf{w}_i \mathbf{w}_i^t}{\|\mathbf{w}_i\|} \mathbf{s}_i}{\sum_{i=1}^{n} \mathbf{s}_i} \tag{5.3}$$

where $\mathbf{s}$ is the separation weighting. This process is quick and requires no searching. Let $\boldsymbol{\lambda}$ be a 1 by n vector of eigenvalues of the EDBFM, $\boldsymbol{\phi}$ be an n by n matrix of normalized eigenvectors of the EDBFM, and γ is the weighted sum described below [38]

$$\gamma_j = \sum_{i=1}^{n} \left|\phi_{j,i}\right| \lambda_j \qquad (5.4)$$

which produces a 1 by n vector of magnitudes representing the saliency of each feature.

5.2.2 Selection. Once the features are rated, there needs to be a selection algorithm to pick which features to use. For most data sets, exhaustive search is not a viable option due to computational constraints. Thus, less exhaustive search methods such as forward or backward selection are used. Forward selection trains a net with each feature individually and picks the best feature based on some criteria, usually classification accuracy. Then a classifier is trained with the best feature and each of the remaining features. The best feature two pair is picked and nets are trained with the best two pair and all the remaining features. This process is repeated until no more features are left. The feature set is selected by which one rated the highest. Ties are broken by the combination that used the least amount of features. Backward selection is just the opposite where features are removed instead of added. These methods are still computationally expensive and do not guarantee the best combinations of features are found.

An alternative selection process first rates the features, then selects them. *Ad hoc* methods simply pick the top x features. Another method is to add a noise feature and pick all features that rate better than the noise [13]. A more systematic method trains a net with the top-rated feature, then with the top two-rated features, and then with the top three-rated features, and so on. This is a rated forward selection which is similar to the forward selection, but it only adds them in the order the features were rated. The set of features used are the ones with the best performance. Although this method does not guarantee the optimal combination, it does coincide with the peaking phenomenon that is often observed and sometimes called the "curse of dimensionality." [14, pp 204] The observation is that as more features are added, classification accuracy increases to a point, where when more features are added accuracies go down. The better the saliency metric the more acute the peaking effect.

MICA-FS selects features by using decision boundary analysis and rated forward selection. However, the problem associated with the rating features by directions of separation is that in multi-class problems the averaging of the outer product of the normal vectors can yield less than useful results. Consider a two space example. If the maximum direction of separation for classes one and two is in the x direction and the maximum direction of separation for separating classes one and three are in the y direction, then when the EDBFM is averaged together the x and y features will be rated equally important. However, on a class pair basis the following is possible:

- Use just feature x to separate classes one and two
- Use just feature y to separate classes one and three
- Use features x and y to separate classes two and three

This approach makes the decision boundary analysis for feature selection more useful. Experimental results have shown this to be the case. For example, a feature has been observed being rated last when separating one pair of classes and being rated first when separating another pair of classes in the same data set. Algorithm 12 shows MICA-FS algorithm.

Algorithm 12 MICA with Feature Selection

```
  $\mathbf{w} \leftarrow []$ {Initialize weight matrix}
  while more class pairs $i$ and $j$ do
    $Class_i \leftarrow ClasseData$ {Transfer all data pertaining to class $i$}
    $Class_j \leftarrow ClasseData$ {Transfer all data pertaining to class $j$}
    $[\mathbf{u}, \mathbf{s}] \leftarrow TrainHiddenNodes\,(Class_i, Class_j)$ {Algorithm 2 slightly modified}
    $\gamma \leftarrow FeatureSaliency\,(\mathbf{u}, \mathbf{s})$ {Combination of Equation 5.3 and Equation 5.4}
    $\mathbf{u} \leftarrow RatedForwardSelection\,(Class_i, Class_j, \gamma, size(\mathbf{u}))$ {Use best features for separating class pairs}
    $\mathbf{w} \leftarrow [\mathbf{w}; \mathbf{u}]$ {Combine new weights in with weights for separating other class pairs}
  end while
```

The Rated Forward Selection algorithm is straight forward, but there are some short cuts employed for faster processing. First, when the feature set is limited to one or only a few features, it is possible for the data in the two classes to be the same value. Therefore the distance matrix, $\mathbf{D}$, will contain zero values. In this case, this feature or combination of features will not be the best combination and therefore another feature is added to the list without testing. Another speedup leverages MICA's ability to add HLNs on the fly. If when using all the features to train the network MICA needs x HLNs, then if a subset of those features needs much more than x HLNs, it is unlikely to result in a better network. Therefore, when MICA is training an MLP with a subset of features and it exceeds $2x$ the number of HLNs as needed by the full features set, then training stops and another feature is added to the list. Once there are enough features to be within $2x$ HLNS of the entire feature set, MICA trains the MLP to completion and then tests it with the test data to get a percent accuracy. The rated forward selection algorithm keeps adding features in the order they are rated until all features have been added. Then the best feature set is the set with the highest test set classification accuracy. The rated forward selection is detailed in Algorithm 13. Note, this approach cannot produce a result worst than using the full feature set because in the worst case the full feature set is used. The rated forward selection algorithm returns a vector the size of the full feature set, but places a zero in the elements of the vector for features that were not used to train the hyperplanes. This maintains the standard MLP architecture. A feature can be eliminated entirely if, in the resulting weight matrix, there is a column of all zero, meaning no class pairs separated using that feature.

Algorithm 13 Rated Forward Selection

```
$\mathbf{TestAccuracy} \leftarrow 0$ {Initialize Test accuracy Vector}
for $i = 1$ to NumFeatures do
    $FeatureIndex_i \leftarrow \gamma_i$
    $Subclass_1 \leftarrow Classe_1(:, FeatureIndex)$ {Transfer columns of features being used}
    $Subclass_2 \leftarrow Classe_2(:, FeatureIndex)$ {Transfer columns of features being used}
    $\mathbf{D} \leftarrow Distance(Subclass_1, Subclass_2)$
    if $min(\mathbf{D}) \neq 0$ then
        $[\mathbf{u}, \mathbf{s}] \leftarrow TrainHiddenNodes(Subclass_1, Subclass_2)$ {Algorithm 2}
        if $size(\mathbf{u}) < maxsize$ then
            $[\mathbf{v}] \leftarrow TrainOutputNodes(\mathbf{u}, Subclass_1, Subclass_2)$
            $\mathbf{TestAccuracy}_i \leftarrow TestMLP(\mathbf{u}; \mathbf{v}, TestSublass_1, TestSubclass_2)$
        end if
    end if
end for
$Index \leftarrow find(max(\mathbf{TestAccuracy}))$
$BestFeatures \leftarrow FeatureIndex(1 : Index)$
$Subclass_1 \leftarrow Classe_1(:, BestFeatures)$ {Transfer columns of features being used}
$Subclass_2 \leftarrow Classe_2(:, BestFeatures)$ {Transfer columns of features being used}
$[\mathbf{u}, \mathbf{s}] \leftarrow TrainHiddenNodes(Subclass_1, Subclass_2)$ {Algorithm 2}
```

5.3 Hidden Node Pruning

HLN pruning can be accomplished in the same manner as feature selection. When feature selection is complete, the data are fed through the HLNs. The HLN outputs are treated as features for training the next layer. If the next layer is another hidden layer, then the algorithm is exactly the same as used for feature selection. If the next layer is the output layer then slight changes are made. In this case, there is only one hyperplane trained per class. Since the saliency metric only involves one hyperplane, the weighting factor is not necessary. The saliency rates the HLNs according to their importance in training an OLN, which turns out to be the same as the magnitude of their weights. Often, many of the HLNs are not needed to training a particular OLN. In a three class problem, some HLNs separate classes one and two, some separate classes one and three, and some separate classes two and three. When training the OLN for class one, the HLNs used to separate classes two and three may not be needed. The same RatedForwardSelection algorithm for the FeatureSelection algorithm is used for the pruning algorithm. However, one additional short cut is used. If the number of HLNs is large, it is not likely that only a small portion will be needed to separate a given class. Therefore a binary search is used to find the transition where x HLNs do not fully separate the class, but $x + 1$ HLNs do. After this point is found, the algorithm precedes as describe in Section 5.2.

5.4 Results

Results are presented for the Iris data and the NIST OCR Data for the following three experiments:

- MICA - MICA
- MICA-FS - MICA with Feature Selection
- MICA-HLNP - MICA with Feature Selection and Hidden Layer Node Pruning

Table 5.1 presents the results for the experiments on the Iris data. These results are the average of 25 test runs. Each test run randomly creates a training and test set and then runs the algorithms. The major result is that the test set accuracy increases and the number of weights decreased almost by half compared to MICA without any reductions.

The training set accuracies did decline slightly for the MICA-HLNP experiment. This is because the algorithm selects the combination of HLNs that have the highest average value between the training and the test set accuracies. It is possible to keep the training set accuracies at 100% by applying additional constraints.

Table 5.1 Iris Data Set Results, 25 Runs

Algorithm	Points	Training	Test	# HLN weights	# OLN Weights
MICA	150	100.00%	95.1%	26.6	18.96
MICA-FS	150	100.00%	96.2%	14.44	16.56
MICA-HLNP	150	99.80%	96.8%	14.44	9.36

The results for the OCR data set are reported in Table 5.2. These results are measured from 5 test runs. The classification results are very close, but the weight reduction is significant. The input weights are reduced by half and the output weights are reduced by more than half.

Table 5.2 OCR Data Set Results, 5 Runs

Algorithm	Points	Training	Test	# HLN weights	# OLN Weights
MICA	3471	100.00%	95.4%	1629.6	592
MICA/FS	3471	100.00%	95.7%	815.8	584
MICA/FS/HLNP	3471	100.00%	95.6%	815.8	267.4

5.5 Conclusions

MICA-FS and MICA-HLNP algorithm are extremely effective at reducing weights in an MLP network, while maintaining or increasing classification accuracies. This ability is afforded by MICA's unique construction style of adding HLNs to separate class pairs. Throughout the development of the MICA algorithm, the Ho-Kashyap method is used to minimize local errors and it results in MLPs those MSE is at the global minimums. Chapter VI addresses the basis for achieving this result.

VI. Theoretical Framework

Chapter II developed the TrainHiddenNodes algorithm, see Algorithm 2, where each HLN was individually minimized. Yet, these local minimizations were effective at minimizing the global error. This chapter analyzes the practicality of performing a series of local minimizations to deliver a global minimization.

6.1 Local Minimization

The Backprop algorithm reduces a global Mean Squared Error (MSE). The reduction in error is caused by the rotation and translation of the hyperplanes from a random start. Backprop moves hyperplanes to areas of high MSE, effecting a reduction of the MSE in that area. Other hyperplanes are then drawn to areas which with a higher MSE. This MSE reduction technique can be remarkable, unfortunately depending on the data set it can also be counter productive. The spiral data set has a low uniform MSE and adversely effects Backprop ability to learn the weights. MICA resolves the spiral problem and yet MICA does not intentionally employ hyperplanes in areas of high MSE. Although dissimilar from Backprop in terms of placement, MICA yields comparable or superior results than Backprop. Hence, placement is a significant aspect of minimization. There is a degree of local minimization in both algorithms ensuing placement. Yet, MICA's learning not only minimizes MSE but maximizes separation as well. Thus, the fitness of the minimization is also a consideration. After all the local minimizations are accomplished, the results are fused to yield a unified result. Consequently, there are three aspects that permit local minimizations to combine for the global effect. They are placement, fitness, and fusion.

6.1.1 Placement.

MICA's hyperplane placement algorithm places hyperplanes between nearest neighbor data points of opposite class. This strategy has two effects; it positions hyperplanes in the locality where there is the greatest potential for effectiveness and as well as locating the hyperplanes in areas of low MSE. This strategy is preferable to Backprop for two reasons. It separates low MSE data that may not be separable with Backprop. Furthermore, the hyperplanes do not incur the problem of local minima as they are being translated through the feature space.

6.1.2 Fitness. MICA minimizes the MSE of a hyperplane and maximizes the number of inter-class data pairs each hyperplane separates. This recasts the problem from one of minimization to one of optimization. Consequently, MICA optimizes the smallest set of hyperplanes possible to separate all the inter-class data pairs of a data set. Unlike MICA, Backprop is strictly a minimization algorithm that inspects all data at once. Thus, Backprop has an opportunity to maximize separation indirectly. Backprop is also confounded by local minima.

6.1.3 Fusion. The products of the independent minimization/optimizations are bound together for a unified result. Since the problem is recast as an optimization problem, the fusing is simply the aggregate of the hyperplanes into a set of lower layer weights. The problem is now a sequential one where once the data are separated, a discriminant function is needed for classification. The result is one hyperplane per class being globally minimized in parallel, whereby the minimization is with respect to separation of classes not classification. Backprop's advantage is that it discerns the impact that changes in the lower layer weights will have on classification. MICA's implementation makes the training of the output layer weights dependent on the training of the lower layer weights, but not vice-versa. For the output nodes to be optimized for maximum separation, the hidden nodes must first be optimized for maximum separation.

6.2 Global Minimization

To attain global minimization necessitates recasting the problem as an optimization problem. The local minimizations/optimizations are bounded by the confines of a global placement algorithm. Thus, each minimization/optimization is sequential and independent in a feed forward manner. Fusion of independent optimizations that are taken in aggregate, compel each optimization to achieve 100% accuracy. That is, the optimization to separate two sets of data are dependent upon each local optimization achieving 100% separation. In general, fusion of sequential events are dependent, but can be optimized independently if done in a feed forward flow. Optimal results are achieved if the first stage is optimized, followed by optimization of the second, then the third, and so on. The optimization of parallel events are independent, but must also achieve 100% accuracies to obtain the global minimization.

Now that a global minimization theory is established, it is applicable to other problems. For example, suppose one endeavored to train a network for function estimation. Then they need an algorithm to situate the hyperplanes in the function space, an algorithm to minimize the MSE of the difference between the hyperplanes and the function data vectors, and finally an algorithm to fuse the hyperplanes together to estimate the function. Of these three algorithms, only the placement algorithm would be a challenge.

6.3 Conclusions

MICA incorporates the ideas of the framework above, and thus achieves a global MSE minimization by minimizing a series of local MSE. The placement algorithm is nominally going to the most difficult and requires the most insight to work out. This dissertation concludes in Chapter VII with conclusions and contributions.

VII. Conclusions and Contributions

7.1 Conclusions

Multiple Layer Perceptrons neural networks serve as a vital tool for engineers and scientists alike. Unfortunately, because Backprop mandates a particular savoir-faire, much of this power is unharvested. Therefore, a multitude of aspiring MLP users either abandon MLPs, or use them and achieve lackluster results. This dissertation presents fundamental innovations in the use of MLP's for classification problems. The results in Section 2.7 substantiate SL-MICA's ability to produce quality results in single hidden layer networks. The results furnished in Section 4.4 establishes ML-MICA's ability to produce quality results in multi hidden layer networks. Section 4.6 illustrates MICA's success at selecting the most advantageous network architecture. The user simply needs to input the data and associated labels and MICA will generate an optimized MLP network that best classifies the data.

Moreover, if a non pattern recognition expert attempts to fashion a MLP they will encounter difficulty selecting the number of features to employ and discerning those features which will produce prime results. Furthermore, an overzealous MLP user may create numerous features with some designed for separating specific classes. The results in Section 5.4 demonstrates MICA's ability to use only the features that optimize performance while separating a specific pair of classes.

In summary, MICA is a great benefit for all users of MLP networks. It will design the complete network architecture, select the best features, and prune unneeded hidden layer nodes to optimize the MLP generalization.

7.2 Contributions

This dissertation introduces a revolutionary MLP construction algorithm that designs and trains MLP neural networks layer by layer starting with the first hidden layer. This algorithm is efficient in designing MLP networks for an extensive range of data sets. It proved itself to be versatile enough to classify any training data to 100% classification accuracy while also optimizing generalization. Additionally, the algorithm designs MLPs that are a mean squared error approximation of the Bayes optimal discriminant function.

This dissertation contributes a feature selection and hidden node pruning technique for use with the construction algorithm above. This technique selects features based on which classes are being separated. This leads to a whole new generation of features that are class-pair separation specific, which when used will not degrade the overall performance of the network. The technique is also applicable to pruning hidden layer nodes to create a MLP network with the fewest number of weights while increasing classification accuracies.

This dissertation contributes the idea of performing local minimizations to achieve a global minimization. There are three aspects in this realization: placement, fitness, and fusion. A framework is development to apply this concept to other problems.

7.3 Follow on Work

There are two areas for additional work. The first is to extend the algorithm to function approximation problems or remain with classification problems but use different separating surfaces . For example, ellipses or hyperbolas can be uses to surround the neighborhoods instead of a hyperplane to simply separate them. Also, Guassians can be placed on top of the neighborhoods to construct a Radial Basis Function Neural Network. The second area is to do more comparison testing with other approaches. The MICA algorithm could be compared against other architecture design approaches. For feature selection, MICA-FS used one saliency metric, but can improve the use of other saliency metrics. These metrics can be testing and compared against the approach in MICA-FS and against traditional approaches where the saliency of the features are rated for the entire problem set.

Bibliography

1. Baum, E. B. "On the Capabilities of Multilayer Preceptrons," *Journel of Complexity*, *4*:193–215 (1988).
2. Baum, E. B. "What Size Net Gives Valid Generalization," *Neural Computation*, *1*:151–160 (1989).
3. Baum, E. B. and D. Haussler. "What size net gives valid generalization," *Neural Computation* (1989).
4. Burgess, N. "A Contructive Algorithm That Converges for Real-Valued Input Patterns," *Journal of Neural Systems*, *5*(1):59–66 (1994).
5. Carter, Martha A. *The Mathematics of Measuring Capabilities of Artificial Neural Networks*. PhD dissertation, Air Force Institute of Technology, WPAFB, OH, June 1995.
6. Cover, Thomas M. "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recongnition," *IEEE Transactions on Electronic Computers*, *22*(3):326–334 (Jun 1965).
7. Cybenko, George. "Approximation by Superposition of Sigmoidal Functions," *Mathematics of Control Signals, and Systems*, *2*:303–314 (1989).
8. Duda, Richard O. and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.
9. Eisenbies, Christopher. *Classification of Ultra High Resolution Radar Using Decision Boundary Analysis*. MS thesis, AFIT/EN/ENG/94D-04, Air Force Institute of Technology, WPAFB, OH, 1994.
10. Fahlman, Scott E. and Christian Lebiere. "The Cascade-Correlation Learning Architecture." *Advances in Neural Information Processing Systems2*, edited by David S.Touretzky. 524–532. San Mateo, CA: Morgan Kaufmann, 1990.
11. Fisher, R. A. "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, *7*:179–188 (1936).
12. Fukunaga, Keinosuke and Donald M. Hummels. "Bayes Error Estimation Using Parzen and k-NN Procedures," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1987).
13. Greene, Kelly A, et al. "Feature Screening Using Signal-To-Noise Ratios," *Submitted to Neuro Computing* (1997).
14. Jain, Anil K. and Jianchang Mao. "Neural Networks and Pattern Recognition." *Computational Intelligence Imitating Life* edited by Jacek M. Zurada, et al., chapter 4, 194–212, IEEE Press, 1994.
15. Larson, Jeffrey E. *Adaptive Neural Network Controller for ATM Traffic*. MS thesis, AFIT/GE/ENG/96D-09, Air Force Institute of Technology, WPAFB, OH, 1996.
16. Lee, Chulhee and David A. Landgrebe. "Decision Boundary Feature Extraction for Nonparametric Classification‘," *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(2):433–444 (March/April 1993).

17. Lee, Chulhee and David A. Landgrebe. "Feature Extraction Based on Decision Boundaries," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(4):388–400 (April 1993).

18. Martin, Curtis E. *Non-Parametric bayes Error Estimation for UHRR Target identification.* MS thesis, Air Force Institute of Technology, WPAFB, OH, December 93.

19. Minsky, M. and O. Selfridge. "Learning in Random Nets." *Information Theory: Forth London Symposium*, edited by C. Cherry. Butterworths Press, 1961.

20. Naylor, Arch W. and George R. Sell. *Linear Operator Theory in Engineering and Science*. Springer-Verlag, 1982.

21. Priddy, Kevin L. *Feature Extraction and Classification of FLIR Imagery Using Relative Locations of Non-Homogeneous Regions with feedforward Neural Networks.* PhD dissertation, Air Force Institute of Technology, WPAFB, OH, April 1992.

22. Reed, R. "Pruning algorithms - a survey," *Neural Networks*, *4*(5):740–747 (Sept 1993).

23. Rogers, Steven K., et al. *An Introduction to Biological and Artificial Neural Networks.* Air Force Institute of Technology, 1990.

24. Rosenblatt. *Principles of Neurodynamics.* New York, NY: Spartan Books, 1959.

25. Roy, Asim, et al. "An Algorithm to Generate Radial Basis Functions (RBF)-Like Nets for Classification Problems," *Neural Networks*, *8*(2):179–201 (1995).

26. Ruck, Dennis W. *Characterization of Multilayer Perceptrons and their Application to Multisensor Automatic Target Detection.* PhD dissertation, Air Force Institute of Technology, WPAFB, OH, December 1990.

27. Ruck, Dennis W. and Steven Rogers. "Feature Selection Using a Multilayer Perceptron," *Journal of Neural Network Computing*, 40–48 (Fall 1990).

28. Ruck, Dennis W. and Steven Rogers. "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function," *IEEE Transactions on Neural Networks*, *1*(2):296–298 (December 1990).

29. Rumelhart, David E., et al. *Learning Internal Representations by Error Propagation, September 1985.* Technical Report, University of California, San Diego: Institute for Cognitive Science, September 1985.

30. Sontag, E. D. "Sigmoids Distinguish more Efficiently than Heaviside," *Neural Computation*, *1*:470–472 (1989).

31. Sontag, E. D. "Feedforward Nets for Interpolation and Classification," *Journal of Computer and Systems Science*, *45*:20–48 (1992).

32. Steppe, Jean M. *Feature and Model Selection in Feedforward Neural Networks.* PhD dissertation, Air Force Institute of Technology, WPAFB, 1994.

33. Steppe, Jean M., et al. "Integrated Feature and Architecture Selection," *IEEE Transactions on Neural Networks* (1995).

34. Tarr, Gregory L. *Multi-Layered Feedforward Neural Networks for Image Segmentation.* PhD dissertation, Air Force Institute of Technology, WPAFB, OH, December 1991.

35. Vapnik, V. N. *Estimation of Dependences Based on Empircal Data.* New York: Springer Verlag, 1982.

36. Werbos, P. *Beyond Regression: New Tools for Prediction abd Analysis in Behavioral Sciences.* PhD dissertation, Harvard University, 1974.

37. Widrow, Bernard and M. A. Lehr. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proceedings of the IEEE*, *78*(9):1415–1442 (September 1990).
38. Wilson, Terry. "Feature Reduction Using a SOFM," *Neural Computing, Submitted* (1997). Submitted.
39. Yang, J., et al. "M-TILING - A Constructive Neural Network Learning Algorithm for Multi-Category Pattern Classification." *In Proceedings of the Worl Congress on Neural Networks 961*. 182–187. San Diego California: INNS Press, September 1996.

Vita

Captain Thomas F. Rathbun was born on January 12, 1964. Capt. Rathbun graduated from Henderson High School, West Chester Pennsylvania, in June 1982. He enrolled in Norwich University the following August to pursue a B.S. in Computer Science Engineering. In May 1986, Capt. Rathbun received his B.S. and was commissioned as a second lieutenant in the Air Force. Capt. Rathbun served as a Test Flight Engineer for the Joint STARS program office before entering the School of Engineering, Air Force Institute of Technology at Wright-Patterson AFB, OH in May 1990. In December 1991 he was awarded the Master of Science degree in Computer Engineering with an emphasis in Artificial Intelligence. Following graduation he was assigned to the Armament Directorate of Wright Laboratory at Eglin AFB, Florida where he did technical program management for the Strategic Defense Initiative. Capt. Rathbun reentered AFIT to pursue a PhD in July of 1994 with an emphasis in Artificial Intelligence and Neural Networks. He is married to Michael Lyn (Evans) Rathbun of Charlotte, North Carolina, we have three children: Sarah, Rhys, and Laurel.

Permanent address: Capt. Thomas F. Rathbun
c/o Davis S. Rathbun
472 Cresent Drive
West Chester, PA 19382

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
June 6, 1997

3. REPORT TYPE AND DATES COVERED
Final Report July, 1994 - June, 1997

4. TITLE AND SUBTITLE
Autonomous Construction of Multi-Layer Perceptron Neural Networks

5. FUNDING NUMBERS

6. AUTHOR(S)
Thomas F. Rathbun

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Wright-Patterson Air Force Base
2950 P Street
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Wright Laboratories
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Distribution unlimited. Available for public release.

12b. DISTRIBUTION CODE
DOD

13. ABSTRACT *(Maximum 200 words)*

The construction of Multi Layer Perceptron (MLP) neural networks for classification is explored. A novel algorithm is developed, the MLP Iterative Construction Algorithm (MICA), that designs the network architecture as it trains the weights of the hidden layer nodes. The architecture can be optimized on training set classification accuracy, whereby it always achieves 100% classification accuracies, or it can be optimized for generalization. The test results for MICA compare favorably with those of backpropagation on some data sets and far surpasses backpropagation on others while requiring less FLOPS to train. Feature selection is enhanced by MICA because it affords the opportunity to select a different set of features to separate each pair of classes. The particular saliency metric explored is based on the effective decision boundary analysis, but it is implemented without having to search for the decision boundaries, making it efficient to implement. The same saliency metric is adapted for pruning hidden layer nodes to optimize performance. The feature selection and hidden node pruning techniques are shown to decrease the number of weights in the network architecture from one half to two thirds while maintaining classification accuracy.

14. SUBJECT TERMS
Autonomous Construction, Dynamic Feature Selection, Neural Networks, Optimized Architechture, Localized Learning.

15. NUMBER OF PAGES
91

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
Unl

Standard Form 298 (Rev. 2-89) (EG)
Prescribed by ANSI Std. 239.18
Designed using Perform Pro, WHS/DIOR, Oct 94